Aspects of Linking CAD and Cost Estimation Software

Yang Liu

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering (Mechanical) at the University of Stellenbosch

Thesis Supervisor: Prof. A.H. Basson

Department of Mechanical Engineering

University of Stellenbosch

December 2001

Declaration

I, Yang Liu, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Abstract

This thesis describes a module that links AutoCAD and CeDeas (cost estimation software which was developed by Department of Mechanical Engineering, University of Stellenbosch). CeDeas is intended for estimating the direct manufacturing cost of simple welded assemblies in a batch production environment. It is aimed at use during late concept design or early detail design.

The link module was developed in Borland C++ Builder. By using COM (Component Object Model) technology, the link module employs the methods and the properties of the AutoCAD automation interface to extract manufacturing information that is required by CeDeas.

The link module prompts the user to pick objects in an AutoCAD drawing and then determines the values required by CeDeas to estimate the manufacturing cost. The user can choose between a "direct select method" (which uses the properties of geometric entities already in the drawing) and a "user define method" (whereby the user defines temporary entities or combines aspects of existing entities in the AutoCAD drawing). With these results and some non-geometric inputs, the user can get a cost estimate of components and assemblies. After design changes, the link module can provide CeDeas with updated values with minimal user interaction in situations where the "direct select method" was used. The designer can therefore easily use the cost estimates to compare design alternatives to optimise the design.

Validation studies demonstrated the numerical accuracy of the use of the link module. The link module can be regarded as an extension of CeDeas. At present it only supports AutoCAD R14, but can be extended to support AutoCAD 2000 and Mechanical Desktop.

Opsomming

'n Module wat dien as skakel tussen AutoCAD and CeDeas (kosteberamingsagteware ontwikkel deur die Departement van Meganiese Ingenieurswese, Universiteit van Stellenbosch) word in hierdie tesis beskryf. Die doel van CeDeas is om die direkte vervaardiginskoste van eenvoudige, gesweisde samestellings, in 'n lot-produksie omgewing, te beraam. Dit is gemik op gebruik tydens laat konsepontwerp en vroeë detailontwerp.

Die skakelmodule is ontwikkel in Borland C++ Builder. Deur van COM (*Component Object Model*) tegnologie gebruik te maak, kry die skakelmodule toegang tot die funksies en eienskappe van AutoCAD se outomatisasie koppelvlak en kan sodoende die vervaardigingsinligting onttrek wat deur CeDeas benodig word.

Die skakelmodule vra die gebruiker om voorwerpe in 'n AutoCAD tekening te kies en bepaal dan die waardes wat deur CeDeas benodig word om die vervaardigingskoste te skat. Die gebruiker kan kies tussen 'n "direkte keuse metode" (wat die eienskappe van geometriese entiteite wat reeds in die tekening is, gebruik) en 'n "gebruiker definieer metode" (waarin die gebruiker tydelike entiteite definieer of kombinasies van aspekte van bestaande entiteite in die AutoCAD tekening gebruik). 'n Koste beraming van komponente of samestellings kan verkry word met hierdie inligting tesame met ander nie-geometriese inligting. Na ontwerpsveranderings, kan die skakelmodule hersiene waardes vir CeDeas voorsien met minimale gebruikers-interaksie in gevalle waar die "direkte keuse metode" gebruik is. Die gebruiker kan daarom maklik die kosteskattings gebruik om ontwerpsalternatiewe te vergelyk om die ontwerp te optimeer.

Evalueringstudies het die numeriese akkuraatheid van die skakelmodule bevesting. Hierdie module kan as 'n uitbreiding van CeDeas beskou word. Tans werk die module slegs met AutoCAD R14, maar dit kan uitgebrei word om met AutoCAD 2000 en Mechanical Desktop te werk.

Acknowledge

The author wish to express his gratitude towards the following person and organizations:

- My parents, Prof. A.H.Basson and NRF for financial assistance.

- My promoter Prof. Anton H. Basson for his enthusiasm, guidance and support through the study.

- My fellow student Kristiaan Schreve and Andreas Schueller for their unselfish support and advice, and Charl Goussard for his help.

- My sisters QingPing Liu, Qi Liu and Cheng Liu, and my friends for their moral support and encouragements.

I thank especially my mother HuiQi Jiang and father YuanYu Liu, who gave me the essential support during my study.

CONTENT

List of Figures

List of Tables

Glossary

IGES	Initial Graphics Exchange Specification
COM	Component Object Model
OLE	Object Linking and Embedding
CE	Concurrent Engineering
B-Rep	Boundary Representation
STEP	Standard for the Exchange of Product Model Data
DFX	Drawing Exchange File
ACSII	American National Standard Code for Information Interchange
DFM	design for manufacture
CSG	Constructive Solid Geometry
API	Application Program Interface
VCL	Visual Component Library
RAD	Rapid Application Development
MFIC	Machinable Feature Interface Component
2D	two-dimensional
3D	three-dimensional
NURBS	Non-Uniform Rational B-spline
AutoLISP	Auto LISt Processing
DLL	dynamic linked libraries
ARX	AutoCAD Runtime eXtension
VBA	Visual Basic Application
BDE	Borland Database Engine
SQL	Structured Query Language
MSDN	Microsoft Developer Network
CeDeas	Cost Estimator in Design Assistant
DCL	Dialog Control Language

CHAPTER 1 INTRODUCTION AND OVERVIEW

1.1 THE ROLE OF COST ESTIMATION IN DESIGN

The need to produce designs that lead to economic manufacturing processes and hence competitive products has encouraged many companies to look for new means to improve product quality, to decrease production cost and to reduce the manufacturing process time of new products. It is widely believed that 70% or more of the life cycle cost of a product is dependent on the early design phases. Cost estimation in this phase plays a very important role in the reduction of product cost. As many design decisions regarding manufacturing aspects are made during early phases, cost estimations will also be very valuable in trade-off studies.

There is therefore an intense interest in developing design tools and aids for on-line cost estimate at early design stages, for users to get quick feedback to optimise their designs. In another words, more effective integration of conceptual design, process planning and cost estimating is required. The generation of a link module to relate the geometric entities of a CAD (Computer Aided Design) application to manufacturing cost estimation software is one of the elements that will help a designer to reach the above-mentioned goals.

During the early phases, the CAD models will typically be incomplete, as manufacturing decisions have to be made before all the finer details of a design can be included in the CAD models. Cost estimation during these phases therefore requires inputs from the designer. These inputs are typically the intended (or candidate) manufacturing processes and quantitative information about the manufacturing features required to obtain a cost estimate. Some of the quantitative information will usually be present in the incomplete CAD model.

However, transferring information between different applications in the neutral data exchange formats, such as IGES (Initial Graphics Exchange Specification), limits the information to the types provided for in the neutral format's specifications. Furthermore, it does not allow for selective updates, for example, when part of a design has been changed and a corresponding update of the manufacturing cost estimate is required. To overcome these limitations, specific software can be developed to interface two or more software packages and facilitate data transfer between the applications.

1.2 THE OBJECTIVES

The objective of this thesis is to develop a user-friendly link module that can relate geometric entities in AutoCAD drawings to manufacturing entities in a cost estimation application—CeDeas (Cost Estimation Design Assistant). The link module should have the ability to obtain geometric information from an AutoCAD drawing and send the information to CeDeas in the form of parameters required to obtain the manufacturing cost estimate. As it stores the manufacturing information in databases, the cost estimation can be updated easily and quickly after the AutoCAD drawing is changed. In other words, it can convert the geometric information in an AutoCAD drawing to manufacturing information and the manufacturing information can be updated easily.

Using a program developed in Borland C++ Builder to communicate with AutoCAD is another feature of this thesis. It aims to investigate another AutoCAD development tool that uses Component Object Model (COM) technology instead of traditional development tools such as AutoLISP, ADS, ARX and VBA.

1.3 OVERVIEW

The link module developed in this thesis involves different aspects of the CAD research field and computer programming technology. In Chapter 2, firstly, a brief overview of CAD is given, feature-based design systems and the feature recognition methodologies are addressed. Secondly, a discussion about AutoCAD application relation to feature-base design is given. Thirdly, COM as programming technology is outlined, and also the development tools and automation in AutoCAD. Finally Chapter 2 gives a description of a cost estimation application named CeDeas and the important characteristics of the link module that interfaces AutoCAD and CeDeas.

A detailed discussion of the implementation of the link module is given in Chapter 3. It includes the programming language, the architecture and structure, modifications to CeDeas, the database of the link module and the main functions of the program.

The evaluation of the link module is done in Chapter 4. It gives the results of the tests of different AutoCAD entities, the case study and some comments of debugging of the link module.

Finally, a conclusion is provided and future work relating to the link module is proposed.

Additionally, the appendices contain a summary of CeDeas and its database structure, the entity types in AutoCAD automation and the main functions of the link module.

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW

A sound understanding of CAD software with respect to engineering design and a computer programming background form the point of departure in the development of an efficient link module as the bridge between CeDeas and AutoCAD. In this chapter, a general review of the pertinent aspects of design process and CAD software in design is provided. Feature-based design is addressed in particular and feature recognition in CAD modules is discussed extensively. Thereafter, an overview of the development tools in AutoCAD and COM technology that connects CAD software to other applications is presented. Finally, the feature-based design aspects of CeDeas and the issues that relate to interfacing CeDeas to AutoCAD are introduced.

2.1 DESIGN PROCESS AND GENERAL COMPUTER-AIDED DESIGN

An overview of the general design process and computer-aided design is given as a basic background.

2.1.1.1 Design Process

One of the difficult, but important, tasks for a design engineer in developing a new product is estimating its production cost during the design process. In brief, the design process is the activity that turns ideas into reality. The design process of a new product can be characterized as an iterative procedure, as shown in Figure 2.1. The four phases shown on the right hand side of Figure 2.1 can be described as follows:

- Clarification of the task involves collecting information about the design requirements and the constraints on the design, and describing these in a specification.
- Conceptual design involves the establishment of the functions to be included in the design, and the identification and development of suitable solutions.

- In the embodiment design phase, the conceptual solution is developed in more detail, problems are resolved and weak aspects are eliminated.

- In the detail design phase, the dimensions, tolerances, materials and form of individual components of the design are specified in detail for subsequent manufacture.

Each of these phases can be split into substeps, and interactions have to be performed between phases in order to optimise the development of the final product.

In order to produce a competitive product, a reduction in cost and lead time and the simultaneous improvement in product quality should stay in the designer's mind during the whole design phase. One of the emerging methods for cost reduction is to involve designers in life cycle cost estimation at the earliest possible stage. This results in a "Design Paradox", as shown in Figure 2.2.

At the beginning of the design process, the designer has much more freedom in his or her design, as few decisions have been made and little cost has been committed to. By the time the product is in production, any change will result in large expenditure, thus limiting the freedom to make changes. Errors made during the early stages of design tend to contribute exponentially to the cost of the final product. For example, an error that costs a thousand dollars to fix in the early design stage may require nearly a million dollars to rectify in the production stage. A rough estimate of the cost can be generated in the conceptual design phase or at the beginning of the embodiment phase, and then, as the product is redefined according to the cost requirements, etc., the cost estimate can be refined as well [Ullman, 1997]. Manufacturing cost estimation that is integrated with the design process will make a significant contribution to realizing this target.

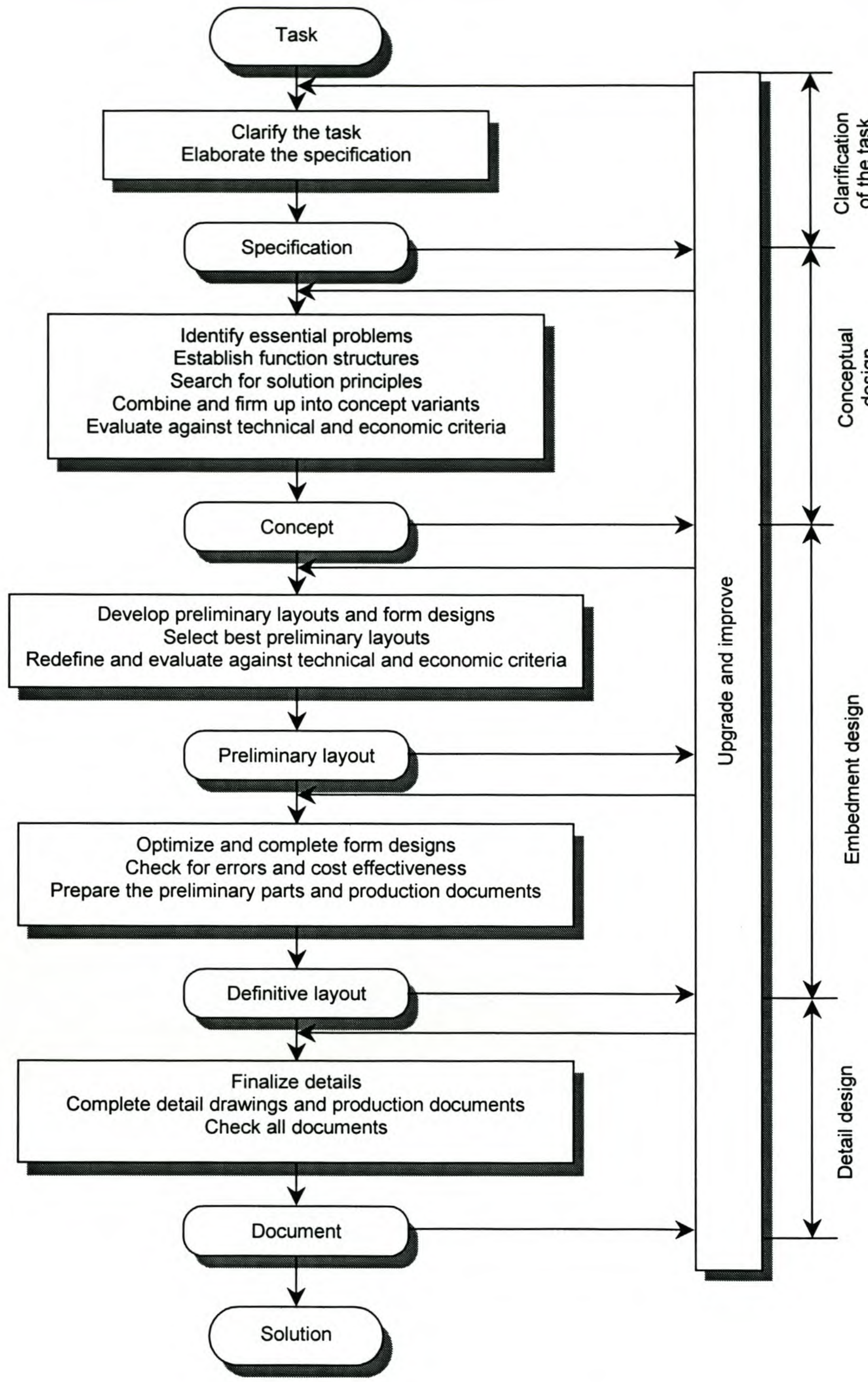


Figure 2.1 The general design process

[McMahon and Browne, 1998]

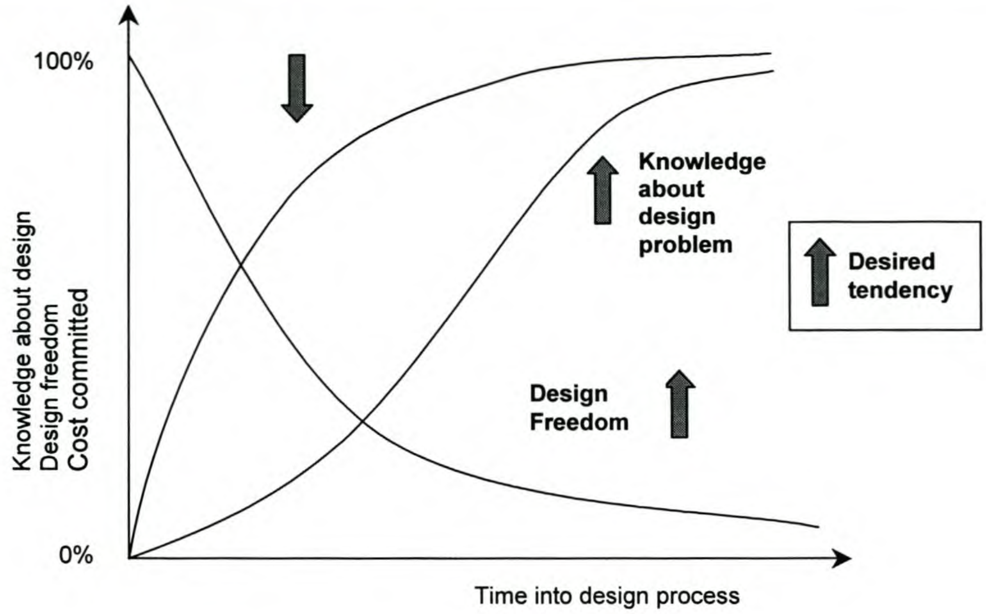


Figure 2.2 Design paradox [Ullman, 1997]

CeDeas is intended to be used in the late conceptual design or early detail design stages to help a designer to obtain a cost estimation of simple welded structure assemblies, that are produced in batches. This will be discussed further in section 2.7.

2.1.1.2 General Computer-Aided Design

Computer software that supports design activities is generally called Computer Aided Design software (CAD application). It can be defined as the use of computer systems to assist in the creation, modification, analysis or optimisation of a design [Singh, 1996]. CAD can be classified in the different ways, according to different researchers:

Based on their constituents, CAD tools can be defined as the intersection of three sets: geometric modelling, computer graphics and design tools [Zeid, 1991]

Regarding the design process as a modelling process, McMahon and Browne [1998] divided CAD into form modelling (such as AutoCAD for engineering sketching, etc.) and system engineering modelling (such as the finite analysis, cost estimation applications, etc.)

According to the different functions for which CAD is applied in the various design stages, CAD is grouped into four functional areas: geometric modelling, engineering analysis, design review and evaluation and automated drafting [Groover and Zimmers, 1984].

These classifications all describe the function of CAD on two levels:

To use a computer to automate or assist in such tasks as the production of drawings and diagrams and the generation of a list of parts in a design.

To provide new techniques that give a designer enhanced facilities to assist in the design process.

One of the major activities in the first level is developing a geometric presentation of a product from the conceptual ideas [Singh, 1996]. It includes computer-aided drafting, computer-aided sketching, etc. But part of the difficulty in geometric interpretation is that the methods traditionally used for the modelling of geometry in CAD are not semantically rich. There is no information, for example, to say that a collection of lines and arcs, or of cylinders and cones, represents a drilled hole and a tapped hole. To overcome this difficulty, the feature-based design technique in CAD can be used to represent product components in terms of higher-level entities named features, which do have some engineering meaning. This can provide a great opportunity for widening the application of CAD and will be described further in the next section.

2.2 FEATURE-BASED DESIGN

In this section, the author will first provide the definition of a feature and then discuss the application of feature-based design.

2.2.1 The Definition of a Feature

It is hard to give the definition of a feature. Most researchers fail in attempting to define the concept of a feature by avoiding being too specific, thus missing some kinds of features. Generally speaking, a feature is any perceived geometric or functional element or property of an object useful in understanding the function, behaviour or performance of that object [McMahon and Browne, 1998]. There are different types of definitions:

Based on the general functions of the feature, a feature can be regarded as a unit that has a semantic meaning in design, process planning, manufacture, cost estimation or other engineering disciplines [Wierda, 1991]. The designer can specify features in engineering terms such as holes, slots, or bosses, rather than in geometric terms such as circles or boxes. Features thus raise the level of abstraction at which the designer may work. Cost analysis for machining is especially suited to this approach, because geometrical features correspond in a direct fashion to the possible machining operations. Holes are related to drilling, slots are related to milling, etc. [Bidarra and Bronsvoort, 2000]. Features can store non-graphic information as well. They allow the user to "tag" relevant data in the CAD database. This information can later be used in activities such as NC machine code generation, finite-element analysis, kinematics analysis, process planning, tolerance analysis and cost estimation.

More abstract than Wierda, [1991], the feature as defined by Bernard and Brissaud [2001] is an element of a part based on a geometrical shape to which technological and topological attributes including quality attributes have been attached. Although these two definitions of features give the general engineering meaning of a feature, different CAD applications use different components of such general definitions and the definition of a feature still needs be more specific in different applications.

According to Gibson et al. [1996], the word “feature” is used at different times for three related but distinct concepts, such as in the feature description language EXPRESS:

- A feature class is a set of feature instances.
- A feature description is the set of criteria whereby a candidate is tested for membership of a feature class, articulated in terms of the components of the candidate and their relationships.
- A feature instance is a structured collection of sub-feature instances, where those sub-features already satisfy criteria and relationships.

Such definitions can be used in feature recognition to find user-defined features within a database of objects of user-defined structure. It is however, still a too general definition to be used in practice.

Pedley and Ehrmann [1995] classify features as application dependent (Table 2.1). They applied these concepts to develop the Simultaneous Engineering System (SESAME), which provides feature-based design, process planning and NC part programming of components typically manufactured in machining centres. Such definitions only address complete geometric data for SESAME. They indicate that the feature is application dependent.

Feature Type	Description
Feature	Object containing information about a discrete component.
Design Feature	Information meaningful in a design context
Design Form Feature	Information representing design geometry
Manufacturing Feature	Information meaningful in a manufacturing context
Dimension Feature	Applied to one or between two elements of the resultant model.
Dimensional Tolerance Feature	+/- Tolerance or ISO limit/fit applied to a dimension feature
Geometric Tolerance Feature	ISO tolerance of position, form or deviation applied to one or between elements of the resultant model.
Surface Roughness Feature	ISO surface roughness tolerance applied to a face element of the resultant model.
Auxiliary Feature	Construction geometry, centre lines, mirror planes and the like.

Table 2.1 Feature definitions [Pedley and Ehrmann, 1995]

Stevens [1992] gives the different definitions based on different viewpoints, as shown in Table 2.2. It is clear that the feature has different meanings relating to the field in which the feature is applied.

Field	Definition
Design	Element used in generating, analysing or evaluating designs
Process Planning	Shapes and technological entities that need to be referenced together
Geometric Modelling	Groups of geometric or topological entities that need to be referenced together
Expert System	Objects formalised by a list of property slots, methods and inheritance features
Database	Groups of associated or related data elements

Table 2.2 Feature definitions in different fields [Stevens, 1992]

According to the issue of application-dependent feature definition, a feature can be defined as follows for CeDeas: Any geometric or non-geometric attributes of a discrete part whose presence or dimensions is relevant to the product or part manufacturing cost calculation.

2.2.2 Feature-Based Design Systems

The definition of a feature forms the base of feature-based design system, which can be classified into two approaches: feature recognition and feature-based modelling.

The intent of feature recognition is to extract information representing a higher conceptual layer than geometry from a set of geometrical information. These are sets of functional meaning within one of the phases involved in the production process. Feature recognition will be discussed in Chapter 2.3.

The intent of the feature-based modelling approach is to provide the designer with a set of functional primitives (design features) to be aggregated and modified to build up a complete representation of that design object. Feature-based modelling usually refers to the construction of geometry as a combination of form features. The basic entity in a feature model is a feature, defined as a representation of shape aspects of a product that can be mapped to a generic shape and that are functionally significant for some product life-cycle phase [McMahon and Browne, 1998].

Boothroyd et al. [1994] favoured a design-with-features approach to determine the costs as the part geometry is built up in the CAD system. However, this may require the availability of a number of process-specific feature-based CAD modules so that the design of a part is achieved through the features, which relate directly to the manufacturing cost contributions for the specific process. On the one hand, the selection of the manufacturing methods is derived from explicit feature information, such as geometry, tolerance and material, and on the other hand, it is derived from the workshop information, such as the available machines and tools.

Hundal and Langholtz [1992] presented a feature-based tool for conceptual design. They use function features to generate the function structure for systems with mechanical, electrical/electronic, fluid and other components. This shows that the application of features goes beyond the construction of geometry.

Gupta and Nau [1995] provided a method for analysing a proposed product design to estimate the manufacturability aspects. By using features that correspond directly with machining operations, it can give the designers feedback on their manufacturing problems. This provides an opportunity to improve the manufacturability characteristics of the product by redesign.

Brett et al. [1996] describe an industry example using feature-based design techniques to specify the relations between different types of holes in one part. By encapsulating access to the individual holes at the level of the radial feature, the designer can alter the number of holes and change the radius of the radial feature to prevent the designer from making inappropriate changes. It extends the capabilities for the user to model relationships between objects, but does not involve determining the feature.

Leibl et al. [1999] developed a feature-based CAD information system. It contains a cost calculation module, a comparison module and a forecast module. According to the degree of detail in the drawings, it uses different modules to determine product cost. By coupling the cost information with the geometry, the designer can obtain cost information during the design process. However, a disadvantage is that the designer must have and use extra data sources that contain features that are manually defined by the user, instead of getting the information directly.

The above examples give the main idea of feature-based modelling. The geometric entities in AutoCAD also have engineering meaning, but they are implicit. The relationship between AutoCAD and feature-based design will be discussed further in section 2.5. The aim of the link module is to get the manufacturing information from AutoCAD and this involves feature recognition, which will be discussed in the next section.

2.3 FEATURE RECOGNITION

In order to perform cost estimation, the designer first needs to obtain the manufacturing features, such as the machining area of a product and the volume of a part, from the CAD model. This procedure may involve using feature recognition technology, which is directly related to the development of the link module.

Feature recognition has been a popular research topic in recent years. In the context of this thesis, feature recognition converts a general CAD model into an application-specific feature model. Feature recognition systems should be able to [Singh, 1996]:

Extract the design information of a part drawn from a CAD database.

Identify surfaces of the part.

Recognize, reason about and/or interpret these surfaces in terms of part features.

Feature recognition research can be divided into three main categories: Feature-class approaches, form feature approaches and graph-based approaches [Gaines et al., 1996]. These three approaches will be discussed further in the following sections.

2.3.1 Feature-Class Approaches

Libraries of features or patterns need to be built to characterize the shapes that are needed to perform analysis with respect to different tasks. The feature searches the part model for geometry matching such feature descriptions. Much research has been done to classify different feature classes or patterns using different methods.

Three classes of core features (internal voids, single and multi-surface holes, and boundary perturbations) from the sets of lower level entities (i.e. the B-Rep structure) were developed by Ganter and Skoglund [1991]. These classes are applied in a feature extraction technique for the generation of casting core patterns from a boundary representation (B-Rep) solid model.

Park and Ko [1996] created a feature library that contains a predefined set of generic features that consist of depression, protrusion and through features that are commonly found in mould manufacturing applications, as shown in Figure 2.3. The system handles 21 elementary features on prismatic components, but the feature set can be expanded. The elementary features are defined on two different levels. The basic class contains field pointers that are commonly needed across all elementary features and the derived parameters that are specific to the feature. The derived class of an elementary feature contains parameters specific to the feature, such as the length and diameter of a hole. They use a core solid modeller to represent the CAD model. It only works on pockets and holes and according to the author, the closure face construction and the edit ability need to be improved.

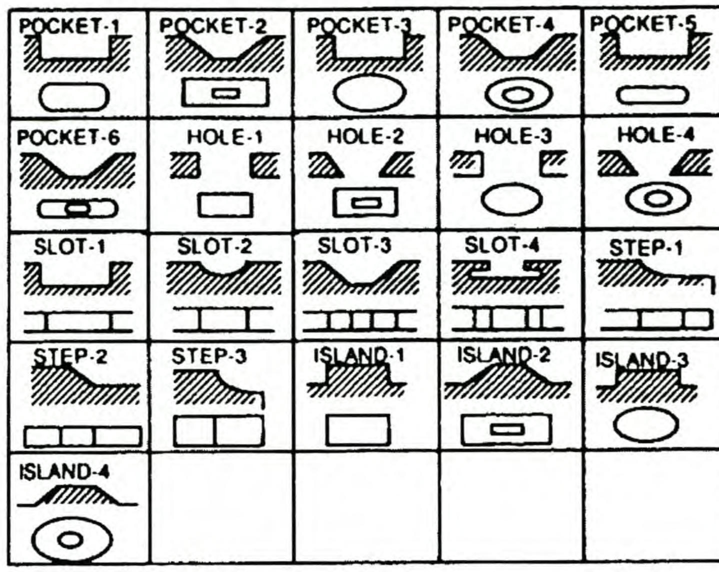


Figure 2.3 Geometry of elementary features for cast mould design

[Park and Ko, 1996]

A feature library (including generic and user-defined features) provided by Hill et al. [1994], which uses Integrated CAD Language (IDL), enables a design to mix cost, material, tolerance and process. The user can add to the existing models in order to enhance their system for design cost/manufacture. According to the author, it can only deal with a limited range of manufacturing processes and industrial environments in detail design.

Esawi and Ashby [1996] have built a database that makes the process selection in mechanical design more systematic. It has three different library classes of product shape features (shown in Table 2.3). The output from its interface is a short list of candidate processes that satisfy the design requirements. It contains a set of members, such as the different types of sub-processes for each process. Each individual sub-process is characterized by a set of attributes, such as size, material, shape etc. that can help the designer to identify possible processes. It is applied in Cambridge Materials Selector (CMS) for manufacture process selection.

Shape feature	Feature classification
Axisymmetric	Uniform
	Stepped
	Dished
Prismatic	Uniform
	Stepped
	Angled
Non-axisymmetric and non-prismatic	Sheet
	Dished
	Complex

Table 2.3 Feature library classification[Esawi and Ashby [1996]

Venuivnod and Yuen [1994] described feature classification in a feature database to provide the topology between features. The taxonomy of features is shown in Figure 2.4 (PTF stands for Primitive/Template Feature and VPTF stands for Variations on PTF). These codes can be used for developing a Multi-Attributed Adjacency Matrix (MAAM) to represent the CAD object. It can handle objects with plane, cylindrical as well as other analytically definable curved faces.

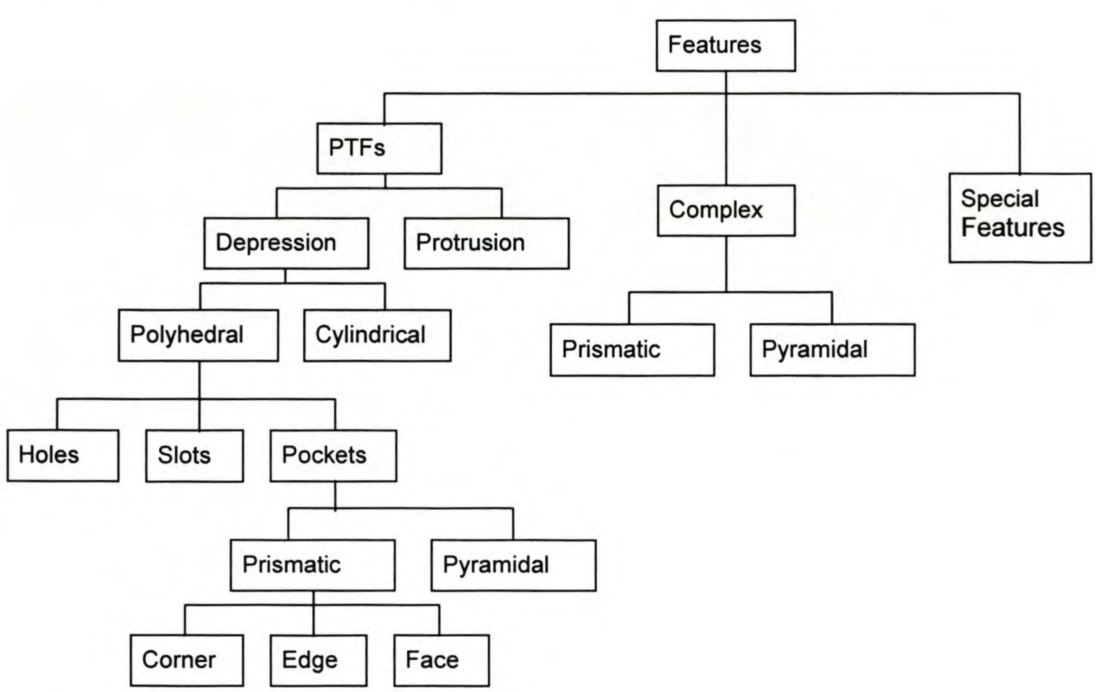


Figure 2.4 Feature code example [Venuivnod and Yuen, 1994]

Pande and Palsule [1988] classified rotational components in CAD models into two groups: external and internal features. These features are sub-classified further into axial, surface and radial types. Each feature is uniquely represented by a three-digit chain code, signifying:

External/internal feature (first digit)

Axial/surface/radial feature (second digit)

Feature identification (third digit)

The three-code hierarchical tree structure for component representation is shown in Figure 2.5. Such features are only available for the rotational components.

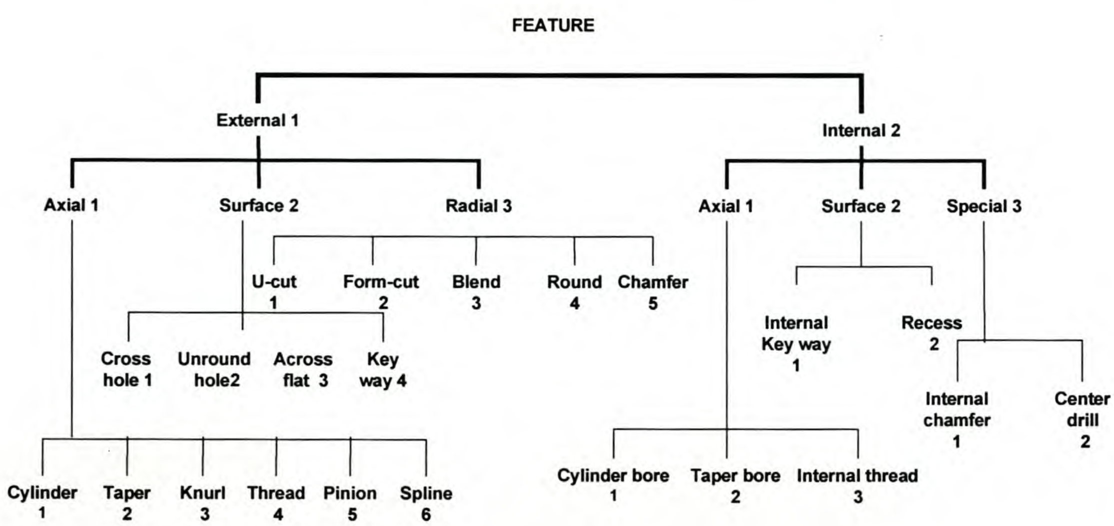


Figure 2.5 Code structure for component representation

[Pande and Palsule, 1988]

Tönshoff et al. [1996] pointed out that each feature instance has two representations: an implicit and an explicit one. The implicit representation includes all information to describe a feature completely and the explicit representation can be derived from the implicit one and contains only the geometric feature data what depend on the used geometric kernel. A feature library was generated from this classification, containing feature schemas that are used as templates during the creation of a feature instance.

Most of these researchers assumed a closed set of design features in the CAD database. However, when a feature that does not belong to the library is generated, the approach cannot handle this feature and it is also not possible to define all possible shapes. The main objective of the link module developed here is to identify the information in an AutoCAD drawing that contains the geometric class instance, and then to form a selection set of the feature class instances to save them in a database. For handling a feature that does not belong to the AutoCAD class library, another option that allows the user to create a new temporary feature is provided in the link module.

2.3.2 Form Feature Approaches

A form feature is defined as a local geometric configuration on a manufactured part that has some engineering significance during the lifetime of the part [Regli and Pratt, 1996]. In other words, it is a geometric formation of an object that can be associated with engineering/manufacturing information. Form feature approaches aim to achieve an increased generality of the feature by avoiding the use of task-specific feature descriptions in order to make a single feature recognition process that is applicable to a broad variety of domains.

Form features can be classified into internal class, modify class and alter class on the basis of their effects on the boundary elements of a basic shape. The internal class form voids inside the basic shape, the modify class modifies the appearance of a basic shape by modifying the appearance of its edges or vertices, and the alter class changes the appearance of a basic shape by becoming incident to its surface, edges or vertices. Qamhiyah et al. [1996] developed a procedure for the sequential extraction of alter class form features from a CAD model of an object with planar surfaces.

By identifying form features, Horváth et al. [1994] have developed a methodology for the generation, representation and manipulation of high level modelling entities called feature-objects. This methodology was implemented in a process oriented feature-objects system. They defined feature objects as the set of application oriented, editable and parameterised feature schemes and classified the features as design feature-object, technological feature-object, inspection feature-object, assembly feature-object and production feature-object for different applications. Such a feature-object oriented approach can generate the accepted form feature paradigm, as it provides tools to map the design information into the CAD database.

By defining features in terms of high-level abstract entities called loops and links, Gadh and Prinz [1992] offered an approach to recognize features and their interactions for a variety of topological and geometric feature interactions.

An approach to perform reasoning about part geometry characteristics for manufacturing assessment was provided by Chen et al. [1992]. It focused on the manufacturability assessment of the discrete parts, with emphasis on the two near net shape processes - die casting and injection moulding. It uses high-level spatial relationships between features ("Is_In" and "Adjacent_To") to support the specialization of the characteristics of individual features, as well as other detailed relationships.

Finger et al. [1992] developed an augmented topology grammar of feature recognition in a computer-based system named Design Fusion. Using such grammar enables a designer to consider concurrently the interactions and tradeoffs among different, and even conflicting, requirements.

Kumara et al. [1994] produced a SRG (Super Relation Graph) method to recognize machinable features in solid models. It provides alternate feature interpretations, which are useful for the optimisation of process planning. The features check the interference of the feature volume and the object, verifying not only the validity of a feature, but also guaranteeing the tool accessibility. Two new relations between faces, super-concavity and face-to-face, were defined as interaction faces in the SRG method. This system is implemented in Borland C++ and integrated into the I-DEAS VI solid modelling system. Gao and Shah [1997] also worked in the field of interaction feature recognition, as did Kumara et al. [1994]. They provided a new approach for automatic recognition of machining features from the B-Rep of the part. This approach combines aspects of graphs and hint-based feature recognition with delta volume decomposition. The program is implemented using the C++ programming language and the ACIS 2.0 solid modeller running on a Unix workstation.

Mahajan et al. [1993] developed a feature-based design system that focuses on a stamping system. The stamping system is based on an integration of design for manufacturing (DFM) and feature-based design. The feature recognition method used in this stamping system is called Virtual Boundaries (VBs), a useful abstraction of a geometric entity, which helps in building a feature in the primary viewpoint. VBs define their own geometries, which are simpler to represent than the geometry of a part as a whole. For example, the narrow faces (forming the periphery) of a regular flat wall can be abstracted as line-segments called edge-faces. The flat wall, which has a uniform thickness, is abstracted at its mid-plane surface and its edges at the corners can be abstracted as corner-edges. Thus, the geometry of a rectangular flat wall is represented as a collection of surface-faces and corner edges. Figure 2.6 shows the different feature classes.

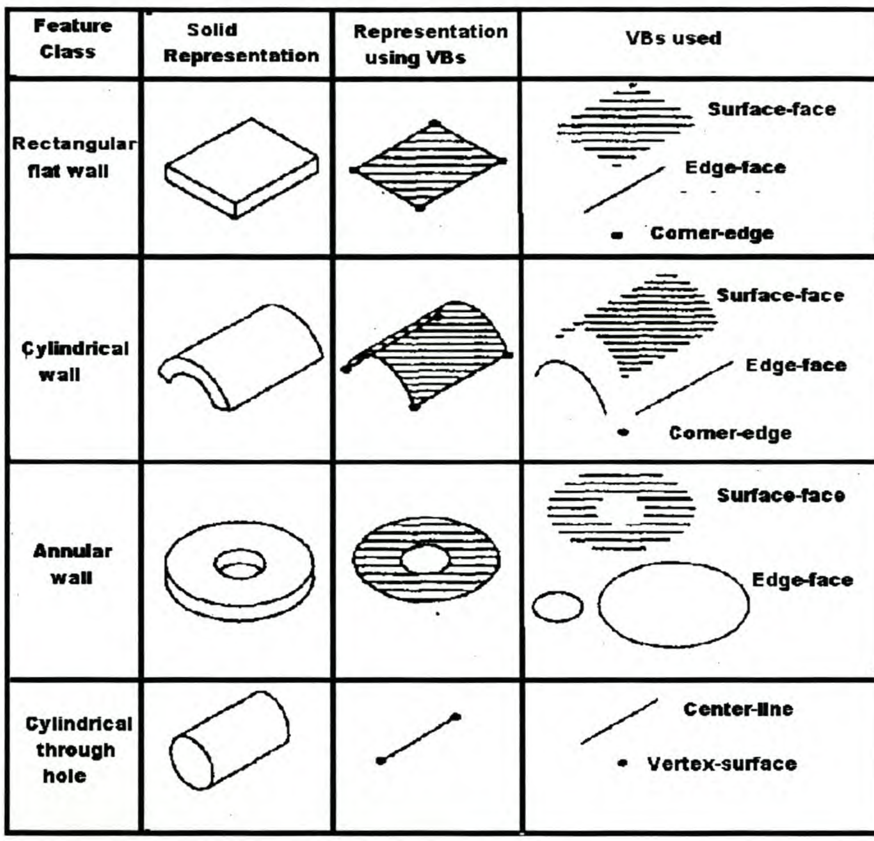


Figure 2.6 Solid geometries and abstracted geometries of some design-with features [Mahajan et al., 1993].

From the analysis of the topology of concave object parts, Stefano [1997] suggested a method for automatic feature extraction from a B-Rep of the solid model in casting product design. This method is based principally on the evaluation of volume and is integrated with visibility map analysis. It works mainly in relation to the casting feature.

Kargas et al. [1988] provided a technique that transforms data and structures corresponding to orthographic views of engineering drawing into a formal, three-dimensional, solid model. The engineering drawing could be generated with a two-dimensional computer-drafting tool. However, it only suits drawings that have only straight lines and circular arcs. The reason for this restriction was that about 40% of the parts designed by a range of mechanical engineering companies could be represented in terms of just two primitives: rectangular blocks and cylinders.

The volume decomposition approach aims to determine the material that was removed from a base part to obtain the required object. Kim and Wang [1999] distinguished machining features into surface features and volumetric features, and decomposed machining removal volume into volumetric machining features. This approach can generate machining precedence relations between features to support machining process planning activities. Through a face pattern based recognition approach, non-interacting volumetric machining features can be recognized.

The difficulty with the form feature approach is that, since form-features are independent of the particular tasks and the context, much post-processing work may be required to translate them into task-specific features that can be used for tasks such as milling, turning or assembly. Essentially, it does not avoid the maintenance problem, but only pushes it into the task-specific post-processor.

In summary, it can be stated that most of the research in feature recognition aims at finding the relations between the different features that can be used for machining planning and generating a new feature. The above-mentioned researchers normally focus on the surface model or solid model and use neutral exchange formats, such as IGES, etc. Krause et al. [2000] suggested that these neutral exchange formats to have the following disadvantages:

Inaccuracy due to transfer

Geometry and topological model inconsistency

Loss of semantic information

Although these researchers provided the various feature recognition approaches to obtain geometric features or manufacturing information from a CAD model, they failed in providing an approach to get geometric information visually from a CAD drawing. These researchers identified the different features from a specific finished CAD model, classified them or represented them in mathematical expressions.

Further, their approaches are normally used for generating new features, process planning, etc. after completing the layout design phase. CeDeas focuses on the stage between the conceptual design phase and completion of the layout design phase.

Thirdly, much post-processing work may be required to translate the particular task independent feature into manufacturing information, and because of the difficulty of maintenance, such an automatic method is not suitable for the link module.

2.4 INTEGRATION OF CAD APPLICATIONS

There is a widespread view that CAD is not adequate as an aid to the designer in generating a design, while CAD systems are applied so extensively that in some companies all the design work is done using CAD systems. CAD is regarded as concentrating too much on providing a means to represent the final form of the design. The integration of CAD systems with other analysis applications is highly needed [McMahon and Browne, 1998].

Recently, Concurrent Engineering (CE) has emerged as a discipline to help to achieve the objectives of cost reduction and the improvement of quality and delivery performance. It can be defined as follows:

> Concurrent Engineering is a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all the elements of the product life cycle from conception to disposal, including quality, cost schedule and user requirements [Singh, 1996].

Concurrent engineering conducts design, development, analysis and the preparation of manufacturing information in parallel. It integrates CAD applications into a CE system to generate a competitive product. A CE environment containing a multidisciplinary, cross-functional team for design is shown in Figure 2.7. It can be seen that the cost estimation (cost accounting) remains one of the basic elements of the CE environment.

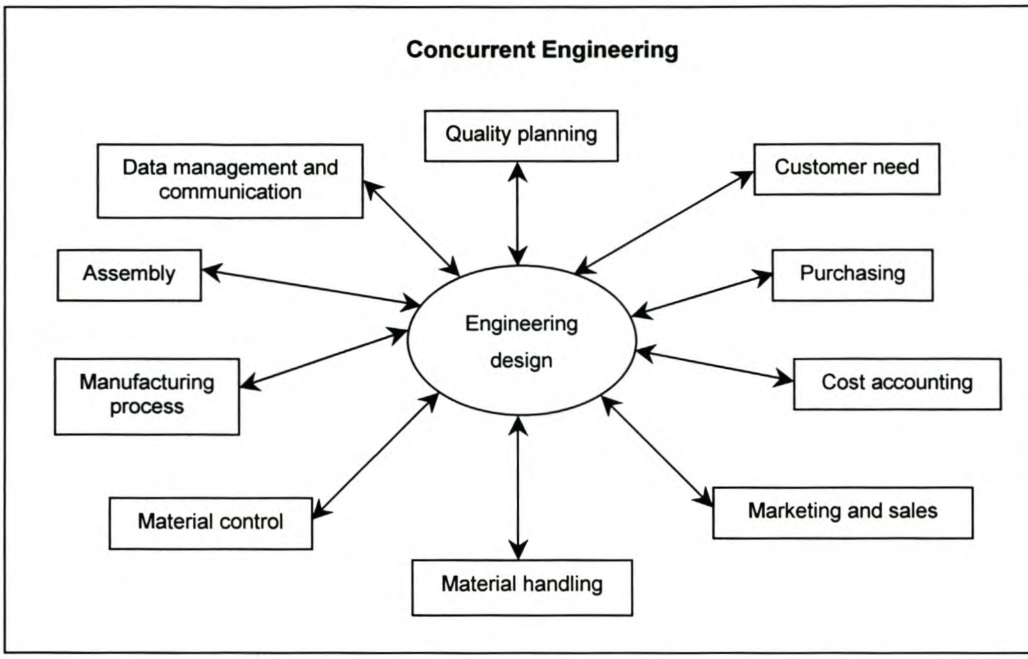


Figure 2.7 Current engineering environment [Singh, 1996]

In order to integrate the design into process planning, Feng et al. [1999] illustrated the communication levels between design and process planning as shown in Figure 2.8. The content includes feature, constraint, geometry, material and manufacturing process.

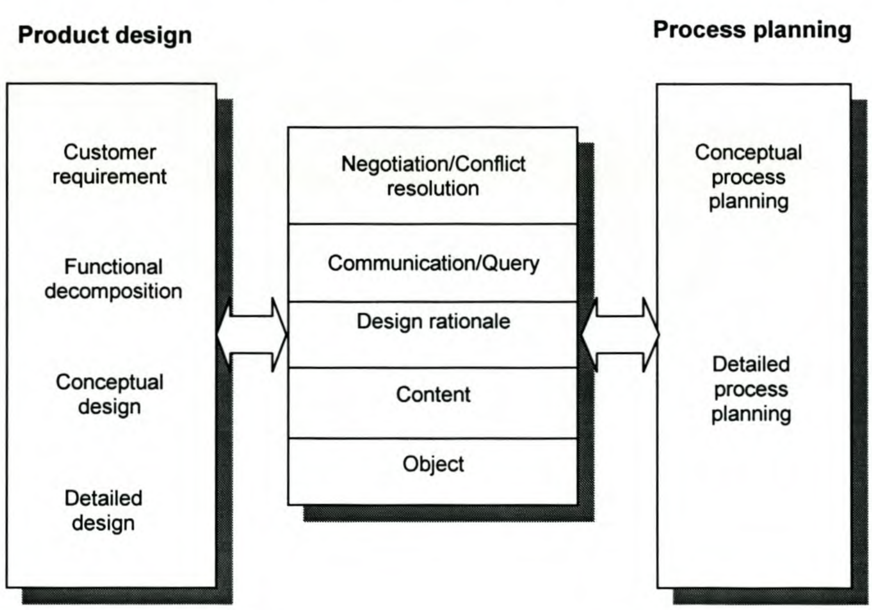


Figure 2.8 Communication levels between design and process planning [Feng et al., 1999]

What follows are several approaches to integrate CAD into other applications at the content level.

Choi and Barash [1985] introduced a sequential and tool oriented process planning (STOPP) methodology that can integrate the entire design-manufacture cycle by providing a direct link between a CAD database and NC code generation. They modelled a cutting tool as a surface generator because a surface shape was generated from the tool profile when it swept through its path. Process planning can be thought of as a mapping from a machined surface to a process plan and they used this concept to develop some process planning functions in PASCAL. These functions are used to generate a Simple Processing Cycle (SPC) for each machined surface in the CAD model, and the SPCs are then converted to produce NC code.

Jayaram and Mylebust [1990] developed an expert system using the IBM Expert System Environment, PASCAL routines and CAD/CAM Interactive Solids Design to automatically generate interfaces between the application programs and the CAD systems. The expert system can create the dimensionally correct solid geometry for different applications. However, the capabilities of this system are only suitable for very simple parts and processes.

Subbaraman et al. [1997] created a new model for on-line cost estimation of a product at an early design phase. By integrating activity-based costing and feature extraction, they produced a module using Activity Based Costing. This model depicts various modules and the components that integrate these modules to achieve the least product cost. Figure 2.9 shows the data flow of this model. They use the form volume decomposition that belongs to the form feature recognition method to ensure that the information relating to a hole, slot etc. from the CAD model is represented in CSG and B-Rep.

Bidanda et al. [1998] developed an expert system for castability analysis and product cost estimation. It provides a dialog interface for data-input customer login, suitability analysis and material selection, the size and weight of the casting, order quantity, geometric features of the parts, etc. This graphic dialog interface provides different class options for each input; the user can thus quickly identify the inputs. The output provides accurate price quotations, including the labour cost, the material cost, the selling cost, etc., and the design time is reduced to several minutes, which had been tested in practice.

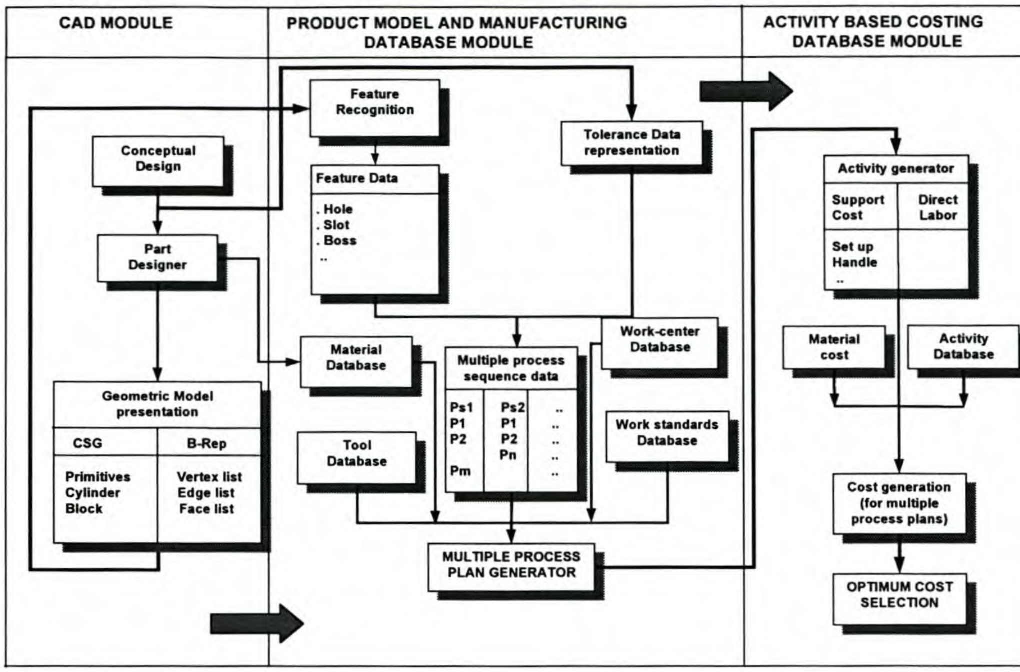


Figure 2.9 Data flow in the model of cost estimate [Subbaraman et al., 1997]

Zhao and Ridgway [1994] have developed a software package named CADEXCATS, which has achieved a full integration between a CAD system and a knowledge based tool selection system. The CADEXCATS package directly accesses IGES files for turning components that were generated in a CAD system, and outputs a work piece representation file in a data format that is compatible with the EX-CATS CAPP system.

In order to manage large volumes of engineering data to define new products, Peng and Trappey [1995] developed an Engineering Data Management System (EDMS). They focused on the development of a user-friendly interface for EDMS to achieve the objective of a user-oriented design and engineering environment. They used the AutoCAD Development System (ADS) and Dialog Communication Language (DCL) as their programming environment and accessed the SQL database management system (DBMS) via AutoCAD SQL Extension (ASE). However, this method is not interactive and is only suitable for standard products.

Working on a geometric tolerancing system with AutoCAD, Tsai and Chang [1996] investigated a CAD-based tool that simplifies the tolerance assignment process and supports geometric dimensioning and tolerancing.

By using a programmatic parametric design approach, Rohm III et al. [2000] created a graphic interface that allows the input of the Cartesian values. An object-oriented program could then create a line of code in many CAD languages, with the values being supplied into the arguments. It provides the flexibility of transferring data into different CAD models, although this procedure focuses only on simple examples and acts as a translator between three CAD languages (CAITA, PRO/E and NSYS).

Cheng et al. [2000] built a Visual C++ interface that can integrate CSG modelling in Pro/ENGINEER and Finite Element Analysis (FEA) for a compressor. The interface can be used for calculating and comparing the data of a scroll compressor with related standards and criteria, and it can cope with simultaneous design activities concurrently within one single environment. It requires that a user inputs the customer requirements, and the interface then performs the thermodynamic and dynamic calculation to get the geometric parameters of the scroll compressor, such as the wall thickness, pitch size, etc. The geometric model can then be analysed by FEA.

Except for that of Peng and Trappey [1995] and Tsai and Chang [1996], the above-mentioned research work does not deal with the AutoCAD software package that is the focus here. Peng and Trappey's [1995] work is used to generate a new feature in AutoCAD by inputting some parameters of the object. It focuses on the parameter design and is suitable for standard products. Tsai and Chang [1996] focus specifically on the tolerances in AutoCAD. Neither of them explains how to interactively communicate with AutoCAD. The other researches use different programming language/environments to develop their applications. No work involving communication with AutoCAD using Borland C++ Builder was found. There is one flexible method that different applications can use to work together in order to enrich the AutoCAD development tools, and that is component technology, which is described inn section 2.6.1.

2.5 AUTOCAD IN RELATION TO FEATURE-BASED DESIGN

There are many commercial, PC-based CAD packages available for assisting in generating engineering drawings. One of the best-known programs is AutoCAD, which was developed by AutoDesk Corporation. It can be used in architecture, mechanical engineering, electrical engineering, electronic engineering and other fields. It is an interactive drawing system designed to permit a user to create, view, manage, plot, share and reuse the information-rich drawings on a graphics display screen.

Mechanical Desktop extends AutoCAD through feature-based modelling to give the user the flexibilities of 3-D modelling. But it lacks the effective development tools in version 2 for the link module developed here. It is difficult to get the manufacturing information from Mechanical Desktop. Consequently, dealing with Mechanical Desktop is not attempted in this thesis.

To get a manufacturing feature from an AutoCAD drawing, it is necessary to know what information the AutoCAD drawing contains, and what geometric information the AutoCAD drawing can represent.

2.5.1 AutoCAD Drawing Function

AutoCAD represents a product in the form of engineering drawings. Engineering drawings are not only the preferred form of data communication for the designer, they are a necessary part of the design process. As the shape of components and assemblies evolve, more formal drawings are used to keep the information organized so that they can easily be communicated to others. Thus, a well-trained engineer has standard drafting skills and the ability to represent concepts that are more abstract and best represented as sketches. Specifically, drawing is used to [Ullman, 1997]:

1. Archive the geometric form of the design.
2. Communicate ideas between designers and between designers and manufacturing personnel.
3. Support analysis. Missing dimensions and tolerances are calculated on the drawing as it is developed.
4. Simulate the operation of the product.
5. Check completeness. As sketches or other drawing are being made, the details left to be designed become apparent to the designer. This, in effect, helps to establish an agenda of design tasks left to accomplish.

6. Act as an extension of the designer's short-term memory. Designers often use sketches to store information they might otherwise forget.
7. Act as a synthesis tool. Sketches and formal drawings allow the piecing together of unconnected ideas to form new concepts.

An AutoCAD drawing contains information that is meaningful, beyond only a composition of different geometries. It can provide some of the manufacturing information that CeDeas needs to calculate the cost. However, this type of information is implicated in the drawing. The representation of such information will be further discussed in Chapter 4.

A significant advantage of AutoCAD is that it enables users to develop their own applications. The ActiveX Automation object model can provide a user with this kind of flexibility. The concept of an AutoCAD application will be presented in Chapter 4.

2.5.2 AutoCAD Geometric Representation

AutoCAD geometric representation is a very important background for this thesis. It can be classified into 2-D and 3-D entities as discussed in the following.

2.5.2.1 AutoCAD Entities

The generation of drawings and diagrams is one of the main functions in CAD systems. It seeks to improve the design modelling process by increasing both the speed with which a design may be represented, and the accuracy of the representation. It achieves this in part by providing semi-automatic facilities for tasks such as the annotation of drawing or for complex constructions, but especially by facilitating the repetitive use of drawing geometry. In doing so it substantially reduces the risk of transcription errors in the propagation of geometry through the design process and, as shall be seen later, in the extraction of geometry for analysis and manufacture.

An AutoCAD drawing is made up of entities. These can be either simple graphic primitives (such as lines, arcs, text, and so on) or blocks (which are groups of entities). The graphic primitives are defined geometrically in terms of the normal Cartesian coordinate system. Hence, for example, lines are defined by their end point coordinates (x, y and z), while circles are defined by their centre coordinates and radii. Each entity also has certain attributes associated with it, such as line style, text font or colour. A block is a group of entities that can be manipulated as a single unit. Once created, a block may be moved, scaled, rotated, copied or deleted.

Entities in AutoCAD can be created by using eight basic item types, i.e. lines, points, arcs, notes, dimensions, arrows, symbols and crosshatching. A user can move, mirror, matrix, rotate or re-scale any individual item, a selected portion of any drawing or an entire drawing. Drawing features can be defined by adding dimensions, notes and crosshatching, and a part listing can be created for each drawing. However, the drawing is not only a collection of these eight items, as it represents more information beyond this collection.

The information in a design can be divided into two types: numerical data specifying length, area, volume, etc., and qualitative information such as notes, identifiers and descriptors of various kinds [Jones, 1992]. All CAD systems model a design as a collection of items (entities), which are mostly familiar geometric objects. In addition, items are included to organize the design by grouping entities together and to provide the descriptive information previously given in notes. Here especially the geometric entities are addressed, and these basic geometry entity types will be illustrated in the following sections.

2.5.2.2 Two-dimensional geometric entities

The AutoCAD system normally contains 2-D (two-dimensional) and 3-D (three-dimensional) systems. Two-dimensional drafting with CAD has traditionally been used by mechanical designers and engineers. For many mechanical design applications, two-dimensional drafting sufficiently and accurately describes the part geometry. Geometric dimensioning and tolerancing, feature control symbols and detailed specifications usually communicate the part or product to the manufacturing engineer or another analysis [Jones, 1992].

Points

Points are fundamental to all AutoCAD software. A line is defined by its start point and end point. Curves are constrained by various points along them. Each point is held as a pair of numbers that are the two co-ordinates in a Cartesian co-ordinate system. However, they do not provide really important information in the final sketches.

Straight lines

A CAD system would not be a CAD system without straight lines [Jones, 1992]. There are two different approaches to generate lines: the software either lets the user create each line separately or makes the user create the lines in sequence as explicit composite lines called polylines. A polyline is a connected sequence of lines or arc segments created as a single object. The polyline allows a user to edit all segments at once, set the width of individual segments, make segments taper, and close the polyline. The information about a straight line in AutoCAD that can be used by CeDeas is the length or the area that is formed by a polyline.

Circle and Arc

Circles and arcs are also fundamental to AutoCAD. Although we can treat a circle as a 360° arc, there may be an advantage in having different data structures and hence different entities. Specifying circles and arcs in different ways allow them to behave in different ways during alterations. For instance, holes typically need to be located by their centres, whereas the adjacent line segments locate fillets.

B-Splines

A spline is a smooth curve passing through or near a given set of points that define the desirable path of the curve. Depending on the mathematical techniques used by the software, the control points are used in a number of ways. AutoCAD uses a particular type of spline known as a Non-Uniform Rational B-spline (NURBS) curve. Two of the control points will usually fix the start and end of a spline. The curve may then be forced to pass through or near the others or be tangent to lines joining parts of them. The control points are thus the means of defining positions through which the curve must go or the direction it must follow at particular places. This should be distinguished from the technique of defining curves in which a large number of points are laid down and the curve fitted to them. It provides a user with the ability to determine what direction the curve should take at any position.

A B-spline can be created by two methods, which are illustrated in Figure 2.10: (A) is the approximating method and (B) is the interpolating method. V(0) and V(n) are the start point and end point, while V(1) to V(n-1) are the control points.

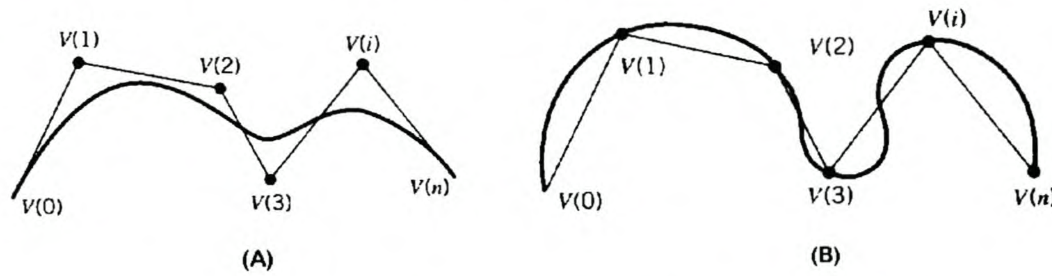


Figure 2.10 Controlling a spline curve [Singh, 1996]

The particular spline generator used in Figure 2.11 constrains the curve to pass through each point. The user may apply an additional direction constraint at each point. The three different splines have been made by simply altering the direction constraints. The ability to apply direction constraints is important for joining two curves together; if the directions of the curves can be made to be the same at the point where they join, the transition from one to the other is smooth, i.e. the direction varies continuously across the joint.

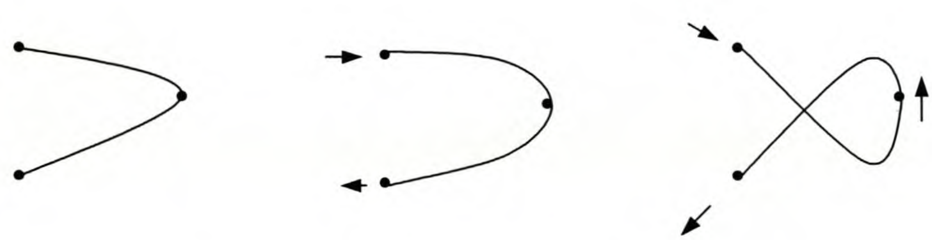


Figure 2.11 Controlling a spline curve [Jones, 1992]

A spline is considered to be a unified representation that can define a variety of curves and surfaces. The premise is that it can represent all wireframe, surface and solid entities. This facilitates conversion from one modelling format to another. However, such an approach has some drawbacks, including the loss of information on simple shapes.

2.5.2.3 Three-dimensional geometric entities

Wire-frame

The first of the 3-D schemes and computationally the most straightforward is the wire-frame geometry. In this, geometry is defined as a series of lines and curves representing the edges of and perhaps sections through the object [McMahon and Browne, 1998].

Wire-frame representation may be regarded as an extension into a third dimension of the technology used for draughting. It is relatively straightforward to use, and is the most economical of the 3-D schemes in terms of computer time and memory requirements. The scheme is particular useful in certain applications involving the visualization of the motion of simple shapes.

Surface

Many of the ambiguities of wire-frame geometric representation are overcome by using the second of the three main 3-D representations—surface geometry. Surface geometry representation involves representing the CAD model by specifying some or all of the surfaces on the component [McMahon and Browne, 1998].

There are a few commonly used surface types in AutoCAD:

Tabulated surface, which is defined by projecting a generating curve along a line or a vector.

Revolved surface, which can be generated by revolving a generating curve about a centreline or vector. It is particular useful for turned parts and axisymmetric parts.

Ruled surface, which is produced by linear interpolation between two different generating edge curves. The effect is of a surface created by moving a straight line with its end points resting on the edge curves.

Edge surface, which is generated by four edges, such as lines, polylines, arcs, circles, etc., and these four edges must form a closed loop.

Solid

An improvement over wire-frame and surface model, both in terms of realism to the user and definition to the computer, is the solid geometry model. The characteristic feature of a solid is that it is bounded by one or more closed surface. Solids are nearly always generated by moving closed lines about in three-dimensional space. A solid is the only entity that fully models the geometry of real objects. It is an unambiguous definition of a closed space whose volume and other properties can be calculated without further definition or resolution of ambiguities by the designer.

There are several common 3-D models and their parameters in AutoCAD are shown in Figure 2.16.

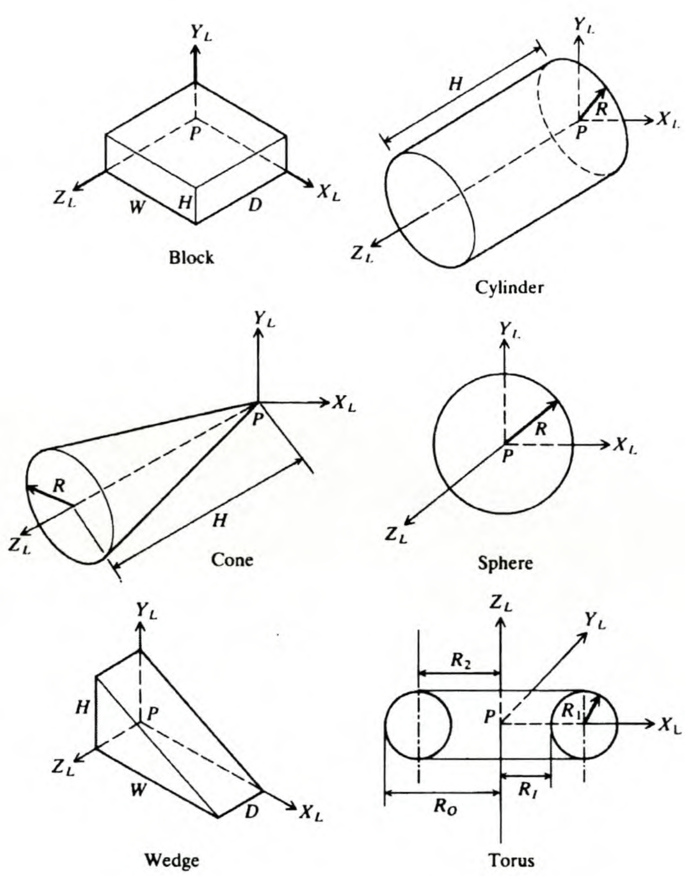


Figure 2.12 Primitives 3-D objects [Jones, 1992]

Based on the primitive 3-D objects, using the Boolean operations of union, intersection and abstractions, as shown in Figure 2.13, can generate complex 3-D solids. In AutoCAD, using these three Boolean operations can generate a new 3-D solid that is a composite of the original 3-D objects. The attributes of the original 3-D objects form a new attribute. For example, each of the original 3-D objects has a volume separately; if they form a new solid, the new 3-D object only has a single volume.

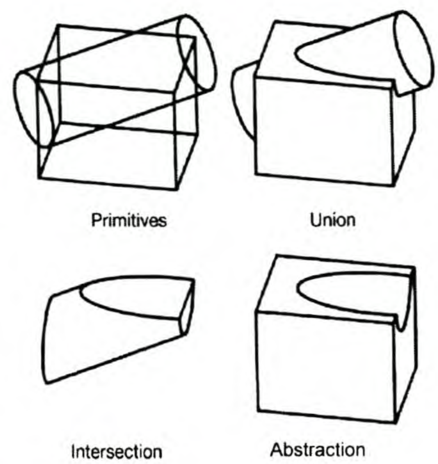


Figure 2.13 Boolean operations of 3-D solid [Jones, 1992]

2.6 PROGRAM INTERFACE

2.6.1 COM

Automatic generation of manufacturing information from a design has been the research focus in advanced CAD/CAM applications, in feature-based applications particularly for design, process planning and other automated manufacturing applications. However, with the current development mode, the functionality and data of one application cannot be accessed by other applications. Monolithic applications are time-consuming to develop and difficult to customize and update to meet the user's requirements. Typically, each researcher independently has to develop a complete set of new monolithic applications built almost from scratch to create features and then perform various process-planning functionalities [Bliznakov et al., 1996]. This development mode leads to such a demanding task in advanced CAD/CAM applications that, although researchers in these fields have developed countless feature-based application programs, the problem of integrating CAD and CAM applications is still largely unsolved.

COM technology, which is changing the computer-programming mode, is a solution to this problem. It provides developers with a new development mode that is much more productive, flexible and powerful than various existing development technologies. For example, it enables developers to greatly simplify and speed up their work by combining or reusing diverse binary "modules" developed independently by various teams and individuals.

In order to introduce this complex programming approach, some concepts are explained in the following sections [Brockschmidt, 1996].

COM

COM is the most widely used component software mode in the programming field. It is an object-oriented programming model for building software applications made up of modular components. COM allows different software modules, written without information about each other, to work together as a single application. COM object creation is language independent. (COM objects can be written in many different programming languages.) It enables software components to access software services provided by other components, regardless of whether they involve local function calls, operating system calls, or network communications. In brief, COM can provide the user with the following abilities:

Writing code that can be used by multiple programming languages

Creating ActiveX controls

Controlling other programs via OLE automation

Communicating with objects or programs on other machines (DCOM—Distributed Component Objects Model)

OLE

OLE is a mechanism that allows users to create and edit documents containing items or objects created by multiple applications. OLE was originally an acronym for Object Linking and Embedding. However, the abbreviation is now referred to simply as OLE. Parts of OLE not related to linking and embedding are now part of Active technology. The structure of OLE technology is shown in Figure 2.14. OLE has two main parts: OLE Documents and OLE Controls.

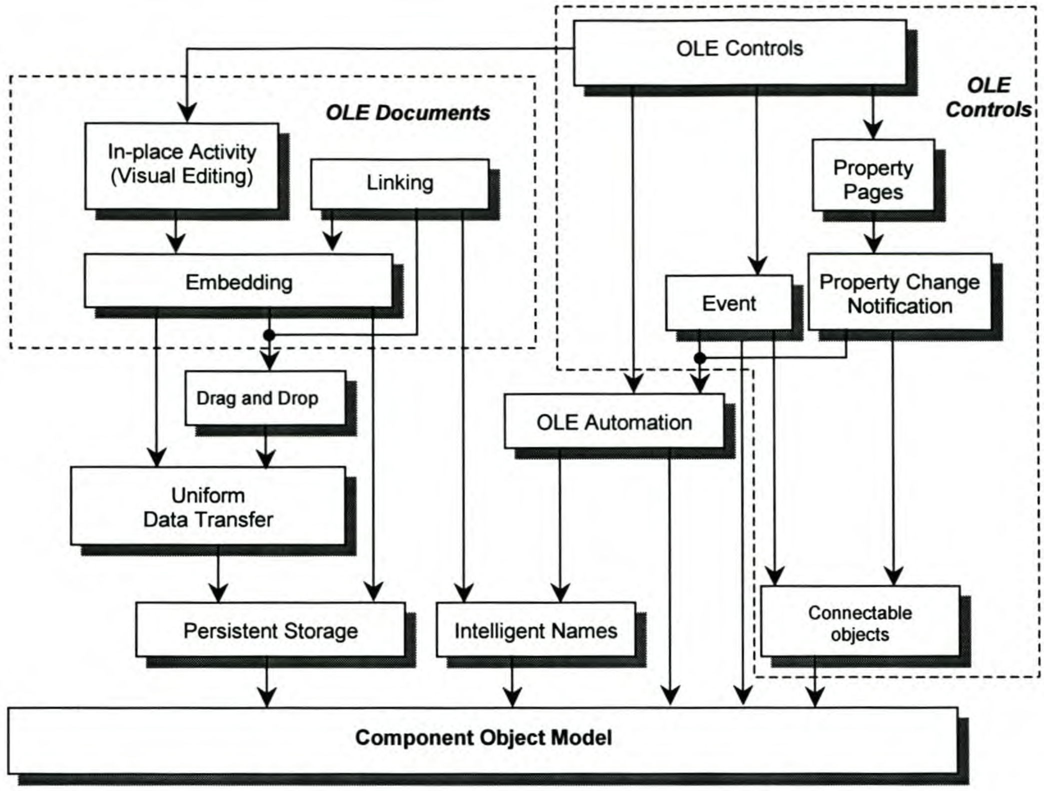


Figure 2.14 OLE technologies build on one another with COM as the foundation [Brockschmidt, 1996]

OLE documents, historically called compound documents, seamlessly integrate various types of data or components. Sound clips, spreadsheets and bitmaps are typical examples of components found in OLE documents. Supporting OLE in an application allows users to use OLE documents without worrying about switching between the different applications; OLE can switch them automatically.

An OLE Control is a compound document object extended with OLE Automation to support properties and methods. It includes a notification named Event, which is fired whenever something happens to the controls, such as a change in state, user input, and so forth. A control is really a device that transforms many different types of external events, such as mouse clicks and keystrokes, into meaningful programmatic events. On the occurrence of these programmatic events, some event handler can execute code, such as showing a button press down and transmitting a character over a modem.

OLE Controls also include property pages, which form a flexible user interface model that any object can use to allow an end user to directly modify its properties. A property page in this technology is easily integrated into a tabbed dialog box along with property pages from other objects, and can to create a consistent and easy-to-use environment for manipulating such data.

Using Borland C++Builder, a programmer can develop all types of OLE objects: servers, containers, automated objects, and automation controllers.

ActiveX and ActiveX control

ActiveX is a component-level technology for building applications from reusable parts; it is a subset of COM. Using ActiveX objects introduces both features and restrictions. These objects can be visual or non-visual; some of them must run in the same process space as their clients; others can run in different processes or on remote machines, as long as the objects provide marshalling support.

The ActiveX control is the new name for programmable elements formerly known variously as OLE Controls or OLE Custom Controls. It is an active Object that may provide interactive or user-controllable functions from within another program or container. The primary difference between an ActiveX control and a COM object is that an ActiveX control has a design-time interface.

Automation

Like other types of objects, COM objects provide methods that their clients can use. Those methods are provided through interfaces that group methods into uniquely named collections. COM objects can choose to expose their methods through two different kinds of interfaces. The first option, called viable interfaces, works very well when the clients that will use those methods are written in C++. The second choice, called dispatch interfaces (usually shortened to dispinterfaces), works very well when clients are programming in simpler languages, such as Visual Basic (although they're also usable from C++ clients). For reasons related to marketing, exposing methods using dispinterfaces has become known as automation.

Automation is a Microsoft technology that allows objects to expose their internal services to each other as well as to human users, thereby permitting tasks to be performed automatically instead of by hand. Automation follows the COM, and most Automation applications derive their objects from the same base interface of COM.

Objects exposed through Automation include ActiveX objects. In general, an ActiveX control is a user interface element created using ActiveX technology.

Relationship between OLE, COM and ActiveX

OLE is the original concept that appeared in 1996. The focus in its first release, OLE 1.0, was solely on the compound documents. COM was introduced in the next release, OLE 2.0, COM grew out of the OLE architect's desire to provide a more general mechanism for allowing one piece of software to provide to be used by another. It is not really tied to compound documents, and COM quickly began to be used in technologies that had nothing to do with compound documents. Unfortunately, Microsoft chose "OLE" as its brand name, which led to the naming confusion. Although Microsoft decided to use "ActiveX" to refer to a loosely defined set of COM-based technologies, "OLE" still refers to compound documents. ActiveX mainly interacts with other components in a networked environment. The development of OLE can be seen in Figure 2.15.

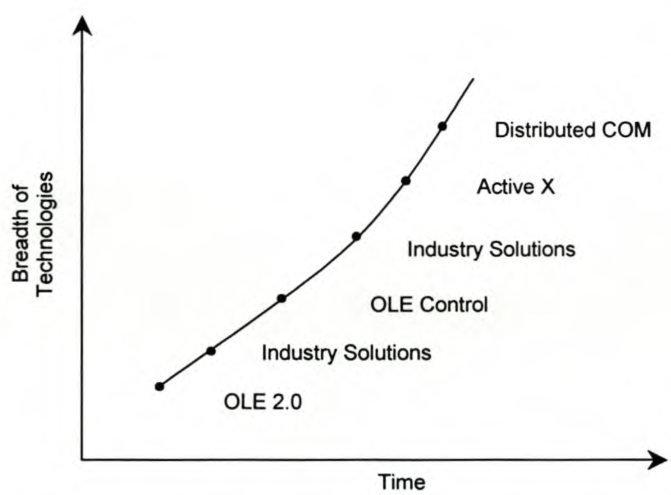


Figure 2.15 The OLE growth curve [Brockschmidt, 1996]

COM is the basis for both OLE and ActiveX. An analogy might be the *TObject* class in C++Builder. All classes in C++Builder are ultimately inherited from *TObject*. Derived classes automatically obtain the properties and methods of *TObject*. They then add their own properties and methods to provide additional functionality. Similarly, OLE and ActiveX are built on top of COM. COM is the foundation for all OLE and ActiveX objects. The COM-based technology is illustrated in Figure 2.16 [Brockschmidt, 1996].

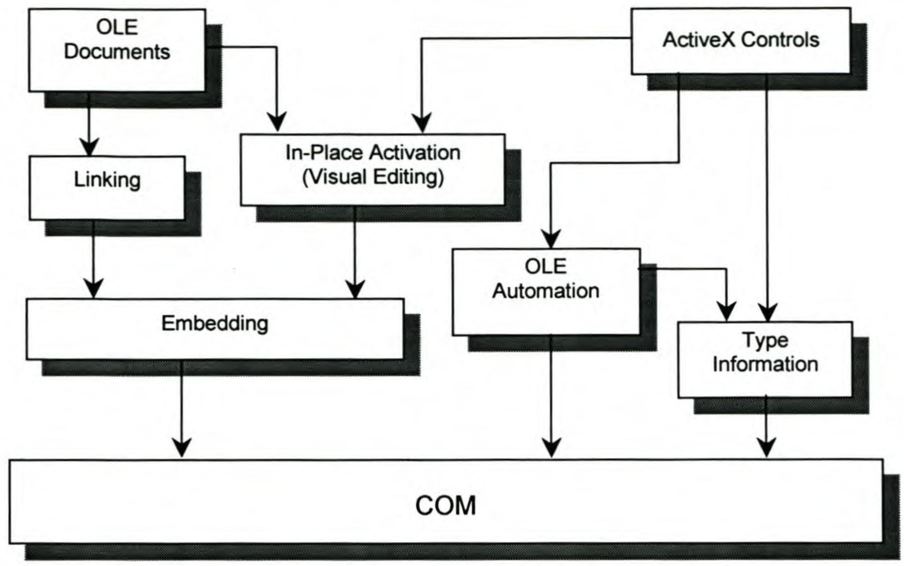


Figure 2.16 COM technology overview [Brockschmidt, 1994]

2.6.2 Component Technology Application

By applying the COM technology mentioned above, a component framework named Advanced CAD/CAM Applications (CFACA) was introduced for feature-based design [Liu, 2000]. It includes mainly two levels: the automation programming interface wrapper and the Application Component Interface Specifications (APCIS). The framework is shown in Figure 2.17.

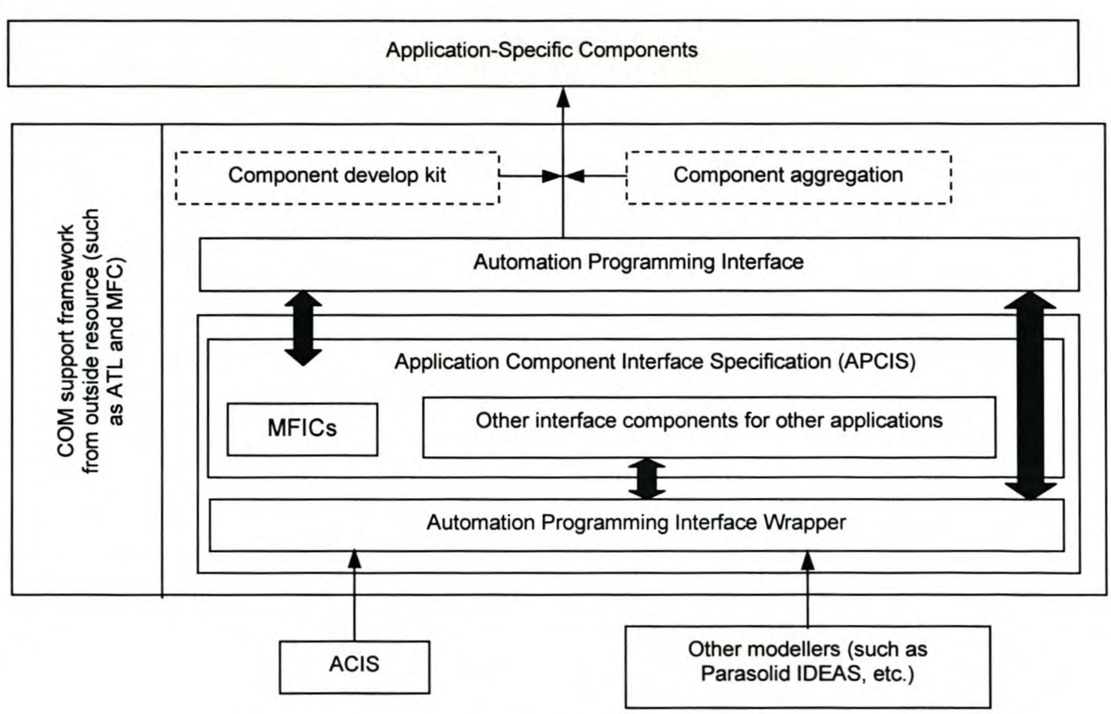


Figure 2.17 Overview of CFACA [Liu, 2000]

The automation programming interface wrapper enables diverse application components to work together and interact seamlessly with one another. This may encapsulate the traditional API (Application Program Interface) through "dual" interfaces, i.e. a dispatch interface and a COM interface. Basically, the geometric objects in geometric modellers, such as a face and an edge, may be "wrapped" or "passed" as the dispatch pointers instead of as the complex traditional geometric pointers. The dispatch interface is an example of the COM binary-compliant interfaces. This type of interface simplifies the development of components with respect to the problems of the marshalling code, languages, the parameter regulation of components, etc. In applications that have the nature of an automation programming interface, the wrapper can automatically and dynamically link geometric modellers. It may also hide the API differences of different geometric modellers. The APCIS deals with accessing or manipulating the geometric information from geometric modellers. They are the common core interface components for the application of specific components, including "Machinable Feature Interface Components" (MFICs), assembly feature interface components, etc. They are the type of interface components that are used for applications tailored to a specific focus area. The APCIS is the common tie among diverse application components from different outside developers in an application domain. Liu [2000] gave an example of a COM application in pocket and hole process planning. Figure 2.18 shows the basic work processes.

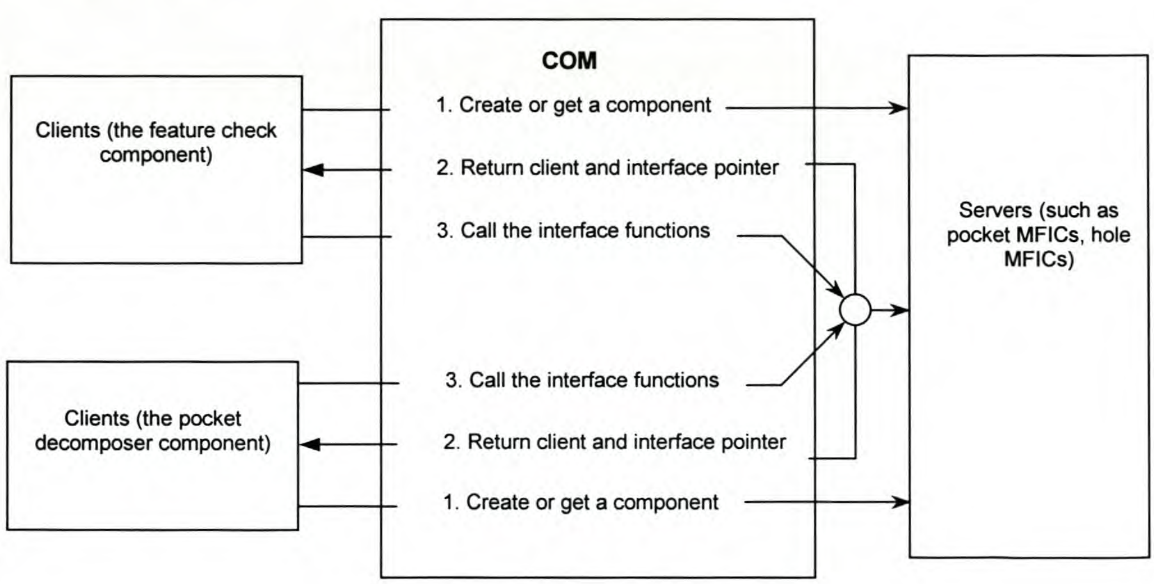


Figure 2.18 The basic work process of an example of process planning [Liu, 2000]

Simply speaking, the components are independent EXE or DLL programs. The component technology provides the means to make a DLL or EXE program (called an automation server) that can expose the functionality to other applications, so that other DLL or EXE programs (called automation clients) can manipulate, share and access the exposed functionality and data belonging to the automation server.

Using COM technology, the CAD system can be developed as a concurrent and collaborative design system that different applications can work with in the same environment, thus reducing the lead-time. This is one of the subjects of this thesis work.

2.6.3 Development Tools in AutoCAD

AutoDesk Corporation has provided several development tools: AutoLISP, ADS, ADS-RX, ARX, ActiveX Automation and VBA. Figure 2.19 shows the history of these development tools.

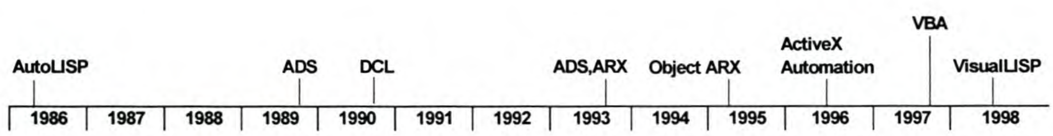


Figure 2.19 The history of AutoCAD tools [Sun et al., 1999]

AutoLISP (Auto LISt Processing)

The first development tool in AutoCAD, AutoLISP, was introduced in 1985. It is an artificial intelligence language and involves symbolic processes. AutoLISP is interactive and safe. The user can change the code without recompiling and it is difficult for its code to crash the computer. However, it is not generally well known and is slow in execution, with the result that it is not suitable to develop large programs.

ADS (AutoCAD Development System)

ADS and DCL became available with AutoCAD Release 11. ADS uses the C programming language, which is well-known and runs faster than AutoLISP, and it is also safe to use ADS to develop applications. However, C is not inherently an "object oriented" programming language and cannot provide the flexibility that C++ does. An ADS application is merely a set of compiled and linked C functions. It is not an independent application and it communicates with AutoCAD only through AutoLISP [Ransen, 1997].

A dialog box in ADS needs to be developed by DCL (Dialog Control Language). DCL can be developed in normal text file format and loaded in AutoCAD to display the dialog box. It includes the description of the dialogue box and the definition of the components of the dialogue box. Almost all dialog boxes are generated by DCL [Xiao et al., 1999].

ADS-RX

ADS-RX is a compiled-language programming environment for developing AutoCAD applications with the ADSRX library of C++ functions. It shares the same memory space as AutoCAD. Most of the functions in ADS-RX are the same as in ADS. ADS-RX became available with AutoCAD R12.

In ADS-RX, applications run as dynamic linked libraries (DLLs) and as a single process with AutoCAD. The dialog boxes used in ADS-RX are also designed by DCL.

ARX

The ARX (AutoCAD Runtime eXtension) was the fourth development tool in AutoCAD. It is called ARX in AutoCAD R13 and Object ARX in R14. ARX was itself written in C++ and the structure of its interface with AutoCAD reflects the idea of communication with AutoCAD "objects" and "entities". ARX is a set of DLLs (Dynamic Link Libraries) and was developed in Microsoft Visual C++ with SDK (Software Development Kit) packages [Li, 1999].

Using ARX, a user can define customized objects and entities from AutoCAD defined standard ones. C++ can access AutoCAD more directly than C can through ADS. This object-based development environment enables developers to create a new generation of intelligent applications for model-based design, where design objects in AutoCAD act as real objects, rather than just as lines, circles and arcs. With the ObjectARX API, professional developers can access and program at the core architecture of AutoCAD and share system resources and memory address space. ObjectARX technology enables applications to extend the AutoCAD drawing database (DWG) at run time, giving developers the ability to add custom objects (entities) to the AutoCAD software's core set of CAD objects.

However, ARX development is not very interactive at run-time. It is more complex and unstable than AutoLISP and C [Ransen, 1997]. The user first needs to compile the program to form a set of DLLs and must then enter the appropriate command to run the ARX code in AutoCAD.

AutoCAD VBA (Visual Basic for Applications)

AutoCAD VBA appeared in 1997 as the fifth development tool. It has the full Visual Basic language syntax, the new forms package, and provides support for ActiveX Controls. VBA first appeared in Microsoft Excel and Microsoft Project in 1994. AutoCAD VBA is an in-process controller, which translates into better performance and throughput in AutoCAD. It also provides application integration with other VBA-enabled applications, which means that AutoCAD, when using object libraries from other applications, can be an automation controller for applications other than AutoCAD.

The integration of VBA into AutoCAD provides an easy-to-use visual tool for customizing AutoCAD. The Visual Basic development environment runs simultaneously with AutoCAD, providing programmatic control of AutoCAD through the ActiveX Automation feature. Using AutoCAD ActiveX Automation and VBA together provides an extremely powerful method for manipulating AutoCAD objects and exchanging data with other applications. The disadvantage of VBA is that it cannot be used to define new AutoCAD entity types by the developer.

Visual LISP

Visual LISP was developed in 1998. It has a Visual LISP development environment and is an extension of AutoLISP. The main module in Visual LISP is an Object ARX application, which can be loaded into the AutoCAD environment [Sun et al., 1999]. Visual LISP can compile LISP code for ARX applications. However, the developer needs to have a background in AutoLISP, which is not provided in AutoCAD R14. Figure 2.20 provides a comparison of these various tools.

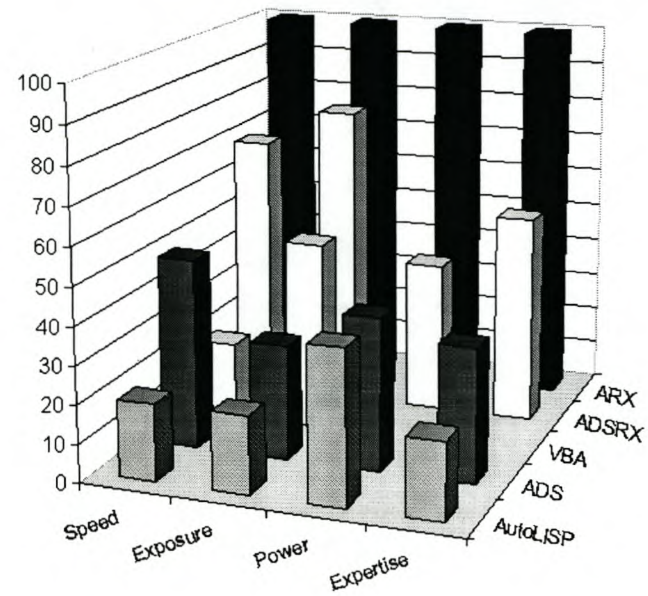


Figure 2.20 Different AutoCAD development tools

[Cottrel and Gao, 2000] & [Sun et al., 1999]

Using an ActiveX object to develop applications is a promising technology for the future. It reflects the trend of AutoCAD tools, as illustrated in Figure 2.21. It can provide the developer with a more powerful and flexible approach to develop different applications.

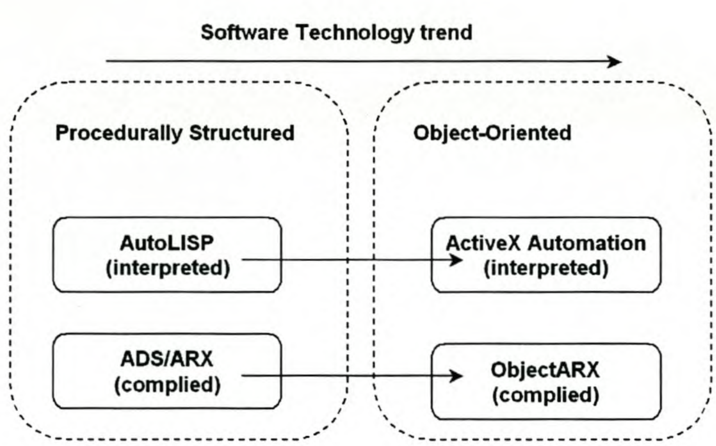


Figure 2.21 The trend towards object-oriented technology

[Cottrel and Gao, 2000]

2.6.4 Automation in AutoCAD

From AutoCAD R14, AutoDesk Corporation provides ActiveX Automation. It enables developers to create a new generation of intelligent applications for model-based design, where the design objects act as real objects, rather than just as lines, circles and arcs. Through these objects, a user can develop applications that are written in different programming languages, such as Visual Basic, C++ and Delphi.

Virtually any type of application can access the exposed Automation objects within AutoCAD. These applications can be stand-alone executables, DLL files, and macros within applications such as Word or Excel. The most common of these is most likely the stand-alone executable.

An object is the main building block of any ActiveX Automation application. Each exposed object represents a precise part of AutoCAD. There are many different types of objects in the AutoCAD interface:

- Graphical objects, such as lines, arcs, text and dimensions, are objects in AutoCAD.
- Style settings, such as linetypes and dimension styles, are objects.
- Organizational structures, such as layers, groups and blocks, are objects.
- The drawing display, such as views and view ports are objects.

Even the drawing and the AutoCAD application itself are considered to be objects.

The objects are structured in a hierarchical fashion, with the Application object at the root. This hierarchical structure is referred to as the Object Model. It is shown in Figure 2.22.

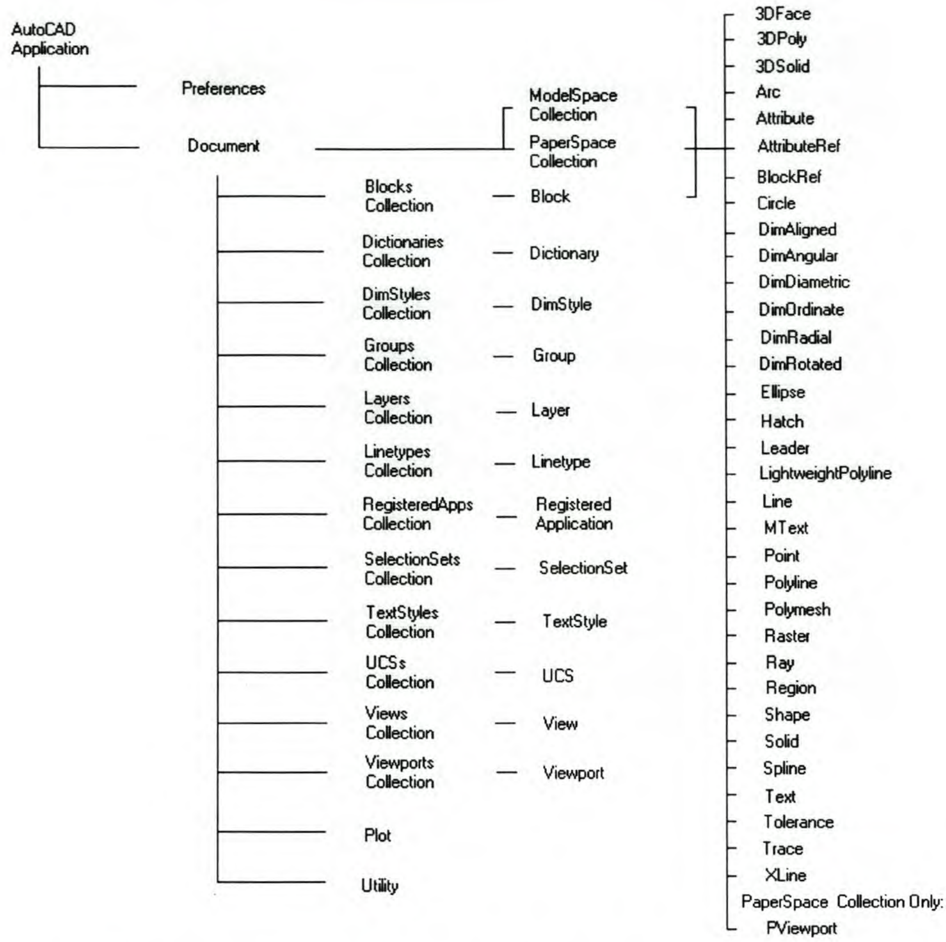


Figure 2.22 The hierarchy of the AutoCAD automation

[AutoCAD on-line help, 1997]

The *Application object* is the root object for the AutoCAD ActiveX Automation Object Model. From the *Application* object, a user can access any of the other objects, or the properties or methods assigned to any object.

The *Document object*, which is actually the current AutoCAD drawing, provides access to all of the graphical and non-graphical AutoCAD objects. These objects are grouped as collections, which allow for easy ordering and processing.

Although collections in an AutoCAD document contain different types of data, they can be processed using similar techniques. The *ModelSpace* and *PaperSpace* collections contain all of the graphical objects found in the drawing's model and paper space. The *ModelSpace* and *PaperSpace* collections have methods and properties for adding graphical entities, extracting a given item, and counting the number of objects in the collection. The non-graphical named objects are found in like-named collections, which also have methods and properties for adding, extracting and counting the items in each collection. The *Blocks* collection contains the list of all *Block* objects (block definitions or inserted entities) in the drawing. Each *Block* object has methods and properties identical to those in the *ModelSpace* and *PaperSpace* collections, for maintaining the graphical objects in each block definition.

The *Document object* also provides access to the *Utility* object, which is a series of methods provided for utility purposes, such as object provides user-input and conversion functions. The user-input functions are methods that prompt the user on the AutoCAD command line for input of various types of data, such as strings, integers, reals, points, and so forth. The conversion functions are methods that operate on AutoCAD-specific data types, such as points and angles, in addition to string and number handling.

Each object has associated *properties* and *methods*. *Properties* describe aspects of the individual object, while *methods* are actions that can be performed on the individual object. Once an object is created, the developer can query and edit the object through its properties and methods. For example, a Circle object has the Centre property. This property represents the 3-D WCS (World Coordinate System) coordinate at the centre of that circle. To change the centre of the circle, simply set this property to the new coordinate. The Circle object also has a method called *Offset*. This method creates a new object at a specified offset distance from the existing circle.

Graphical objects, also known as entities, are the visible objects (lines, circles, raster images, and so forth) that make up a drawing. To modify or query these objects, it needs the methods or properties of the objects themselves. Each graphical object has methods that allow an application to perform most of the AutoCAD editing commands, such as Copy, Erase, Move, Mirror, and so forth. These objects also have methods for highlighting and updating, as well as retrieving the bounding box of the object. Some of the important typical properties and methods are the following:

EntityName and EntityType property

The *entity name* is equivalent to the class name of the object. It is indicated as a string in AutoCAD automation.

The EntityType property is used to get the type of entity, such as *acLine*, *acPloyLine* and *acSolid* etc. All drawing objects have their own Entity Names and Entity types.

Handle property

Handle is an object identifier that used for referencing an object. A handle is persistent (stays the same) in a drawing for the lifetime of the object. Each drawing object has its unique *Handle*, so it is easy to distinguish between different objects in the drawing by using their *Handle* properties.

Add method

All collection objects have this method. It creates a member object and adds it to the appropriate collection. Once added to the collection, the properties of the object can be set using the "object property" syntax.

Highlight method

All drawing objects and selectionset objects have this method. It sets the highlight status for the given object, or for all objects in a given selection set.

Item method

All collection, group and selectionset objects have this method. It can be used to get the member object at a given index in a collection, group or selection set.

SelectionSet collection

It is the collection of all selection sets in the active drawing. *SelectionSet* is a group of one or more AutoCAD objects specified for processing as a single unit. The user can use different methods, such as *SelectAll* and *SelectOnScreen,* to get the Select object from an AutoCAD drawing.

Graphical objects also have specific properties, depending on their object type. For example, a line has StartPoint and EndPoint properties, a circle has CenterPoint and Radius properties and a 3-DSolid has the Volume property.

Non-graphical objects are the invisible (informational) objects that are part of a drawing, such as layers, linetypes, dimstyles, selectionsets, and so forth. To create these objects, use the *Add* method of the parent collection object. To modify or query these objects, use the methods or properties of the object itself. Each non-graphical object has methods and properties specific to its purpose.

One of the main benefits of automation is that the programmer can treat the entity in AutoCAD as an object, which has methods and properties that the user can use or get directly without understanding the internal mechanism. Programming with automation is simpler than programming with ARX, because users do not need to structure a complex class themselves and, most importantly, users can develop an application that can access the entity in AutoCAD from outside of AutoCAD. It provides developers with more flexible ways to build their applications.

2.6.5 Accessing AutoCAD Using VBA

There are several different types of Microsoft's Visual Basic available for use with AutoCAD ActiveX Automation. The type that is distributed with Release14 of AutoCAD is Microsoft's VBA.

As mentioned before, VBA was designed to provide rich development capabilities. Implemented in AutoCAD, these capabilities are aimed at shortening the development times of custom business solutions.

VBA has full Visual Basic language syntax, contains the new forms package, and provides support for ActiveX Controls.

The stand-alone development editions of Visual Basic, which must be purchased separately, complement AutoCAD VBA with additional components, such as an external database engine and report-writing capabilities.

VBA is an in-process controller, which translates to better application performance and throughput for AutoCAD Release 14. AutoCAD VBA shares AutoCAD's memory and process space, thereby improving performance significantly. It also provides application integration with other VBA-enabled applications. AutoCAD, when using object libraries from other applications, can be an Automation controller for applications other than AutoCAD.

When AutoCAD is being controlled by an external VBA-hosted application (such as Microsoft Excel) or by a Visual Basic program, performance is compromised. The external application must communicate with AutoCAD through the Windows message loop using remote procedure calls (RPC). Using VBA as ActiveX Automation controller delivers performance levels surpassed only by native ObjectARX-compiled executables.

VBA sends messages to AutoCAD by means of the AutoCAD ActiveX Automation Interface. AutoCAD VBA permits the VBA environment to run simultaneously with AutoCAD and provides programmatic control of AutoCAD through the ActiveX Automation Interface. This coupling of AutoCAD, ActiveX Automation and VBA provides an extremely powerful interface not only for manipulating AutoCAD objects, but also for sending data to or retrieving data from other applications.

There are three fundamental elements that define VBA programming in AutoCAD: The first is AutoCAD itself, which has a rich set of objects that encapsulate AutoCAD entities, data and commands. The second element is the AutoCAD ActiveX Automation Interface, which establishes messages (communication) with AutoCAD objects. Programming in VBA requires a fundamental understanding of ActiveX Automation. The third element that defines VBA programming is VBA itself, which has its own set of objects, keywords, constants and so forth, which provide program flow, control, debugging and execution. On the basis of an understanding of the VBA syntax and ActiveX automation, a developer can use COM technology via VBA to access AutoCAD entities using other languages. The details of the implementation of this approach are given in Chapter 5.

2.7 CEDEAS

Experienced designers are usually able to create successful initial designs because of their in-depth knowledge of best practices, customer expectations and manufacturing processes. However, less experienced designers often require inputs from experienced designers in all of these areas [Feng et al., 1999]. An expert system that facilitates manufacturing cost estimation at the early stages of design can help to solve this problem. It can also help the experienced designer to overcome some of the limitations of relying only on design experience.

To fulfil this task, a software tool called CeDeas was developed by the Department of Mechanical Engineering, at the University of Stellenbosch. It is an expert system that forms part of a software tool named Design Assistant. It was initially developed by Schuster [1997] and included cost estimation models for welding assemblies produced in batches that were developed by Schreve [1997] and Maree [1997]. The present implementation of this software focuses on the simple, weld structures in a batch production environment, as well as on the following processes: assembly, welding, guillotine cutting, sawing, flame cutting, bending, rolling, punching and drilling.

2.7.1 Overview of CeDeas

CeDeas can give the user a cost estimate of a component or assembly during the late conceptual design or the early detail design stage. The user can obtain the material cost, machining cost, each part/assembly is cost, total cost of the component or assembly, etc. Through various dialog boxes, the user must input manufacturing information to get the cost of different components and the total cost of the project. The designer can make a comparison with alternative concept designs so as to optimise his/her design.

Another function of CeDeas is cost analysis. The designer can be made aware of the impact on manufacturing cost of his/her design decisions and can then try alternatives, e.g. changing the part geometry or using different manufacturing process, etc. Figure 2.23 shows the steps of this procedure.

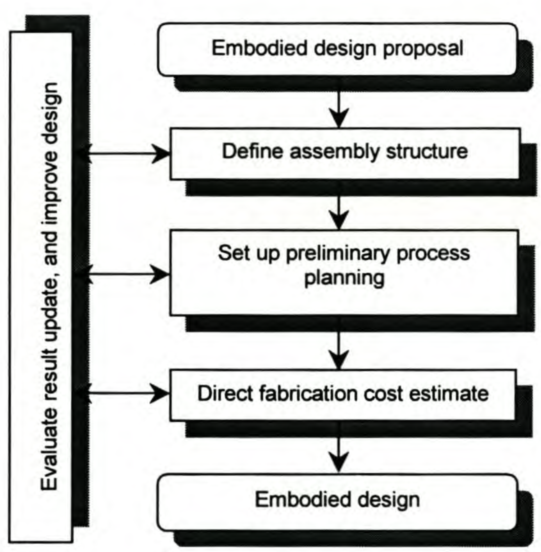


Figure 2.23 Main steps of CeDeas

In CeDeas, a project and its components is regarded as nodes in different parent-child relations. Figure 2.24 shows the structure of a project in CeDeas. When starting with CeDeas, the user must first enter a project and define a bill of materials. Through a tree view, shown in Figure 2.25, the user can define eight different node types: projects, assemblies, parts, library items, outsourced components, processes, set-up elements and operation elements. For projects, assemblies, parts, library items and outsourced components, the user must input the name, description, drawing number and quantity, etc. Each manufacturing process is divided into elements. This step is required to allow the designer to "fine-tune" the cost estimation and to check the influence of some detail design decisions. For some elements, the designer will be requested to enter data for the particular part or process features, such as the length of the part, the machine power, etc. The manner in which this procedure was modified to accommodate the link module will be described in Chapter 5.

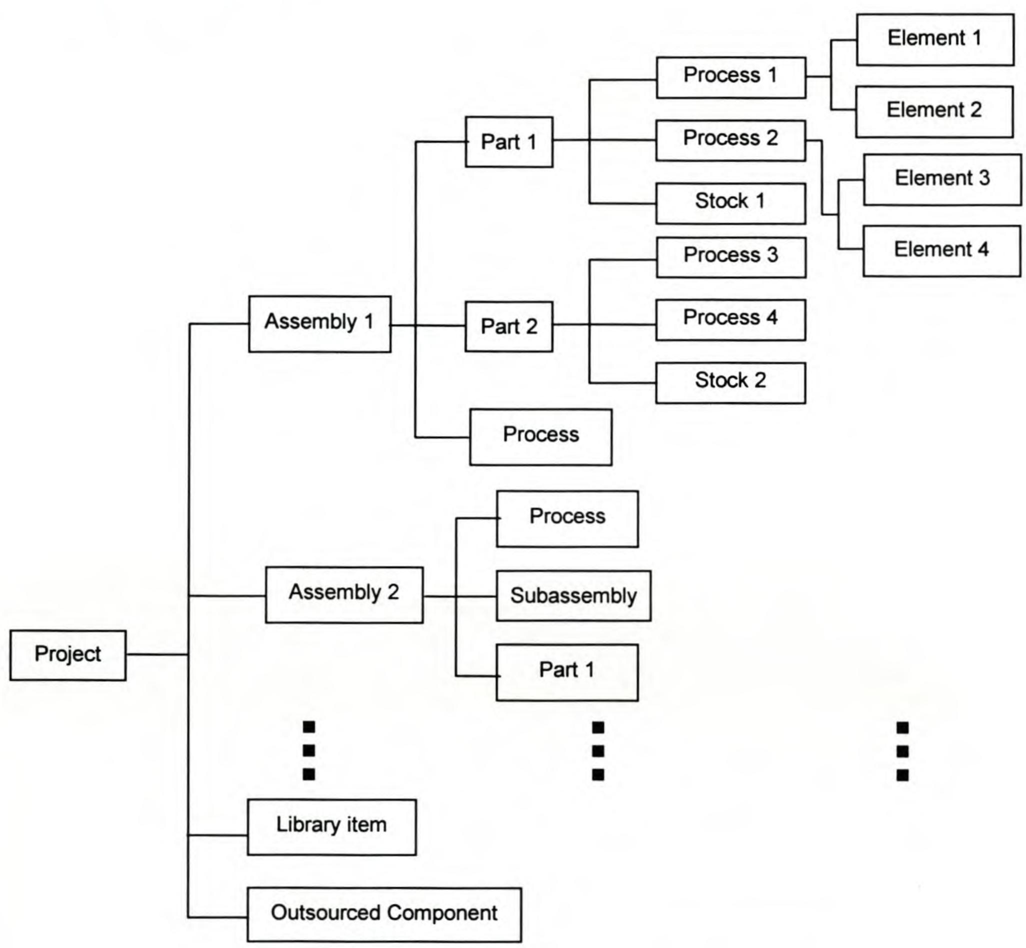


Figure 2.24 Structure of a project in CeDeas

After completing the data entry, the software shows the designer the manufacturing cost make-up of the design, proving easy identification of areas where redesign can have the greatest benefit. The influence of batch size and make/buy decisions can be accessed easily. All of the data entered by the designer can easily be changed, thus allowing convenient evaluation of alternative designs and processes.

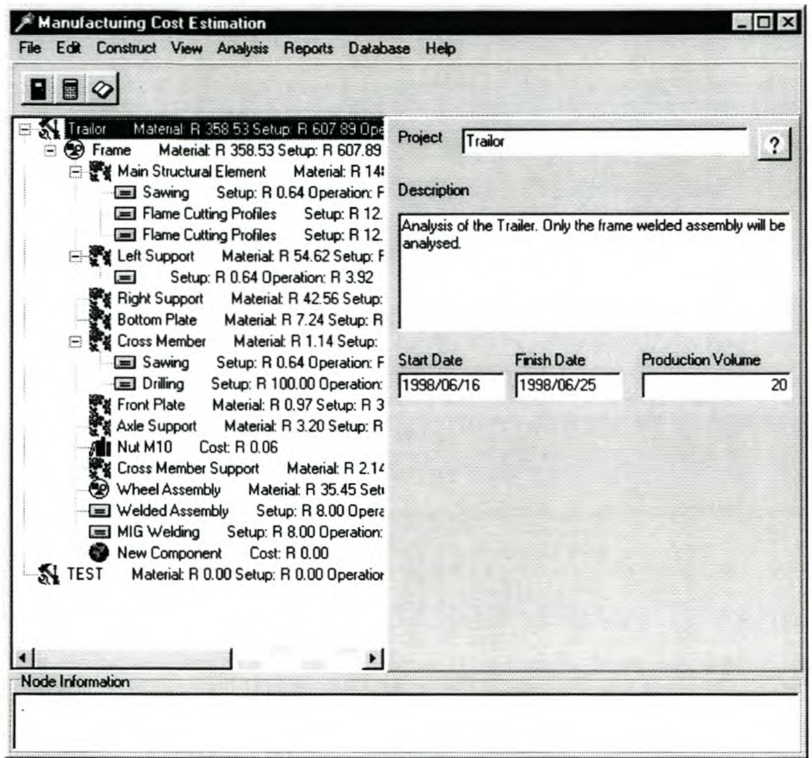


Figure 2.25 User interface of CeDeas

A more detailed description of CeDeas is given in Appendix A.

2.7.2 Relation to Features

In order to calculate/estimate the cost of a part or an assembly and then of a project, the manufacturing features of the part and assembly must to be entered into CeDeas as the cost calculation parameters. Some of them can be obtained directly from the part dimensions, such as the length of a part. However, some of them are contained implicitly in the part/assembly drawing, such as the area of the part. These inputs can be regards as features, since the several lines that form the edge of an area can be considered as an area feature in the AutoCAD drawing. To be able to interface AutoCAD to CeDeas, the approach in this thesis is to use the feature recognition methodology mentioned in section 2.3 to convert the geometric representation into manufacturing information.

2.7.3 Transmitting the Manufacturing Features to CeDeas

In order to get the cost of a project, assembly and part, the designer needs to enter many kinds of manufacturing information relating to the product. This includes the length, volume, angle and projected area, etc.

Different materials and processes in CeDeas need different manufacturing information. The following is a list of the input information required for the different manufacturing processes:

1. Material selection

1.1 For profile option

Part length

Part volume

1.2 For sheet metal option

Part length

Part width

Projected area

2. Process selection

2.1 Punching process

In loading and unloading time item:

Projected area (A)

In punching cycle time item:

The distance between different holes (L_{holes})

In tools change and reorientation item:

Maximum distance between holes (L_{Hmax})

Machine bed length (L_{bed})

2.2 Bending process

No information needed from drawing

2.3 Guillotine cutting process

In guillotine operation item:

Sheet projected area (A)

Sheet length (L_s)

Sheet weight (W_s)

2.4 Sawing process

In sawing operation item:

Profile part length (L)

Profile part volume (V)

Cross section area (A_{cross})

Stock length (L_{stock})

2.5 MIG welding process

No information needed from drawing

2.6 Rolling process

In rolling operation time item:

The length that needs to be rolled (L_{roll})

2.7 Flame cutting sheets process:

In loading time item:

Sheet Length (L_s)

Width of Sheet (W_s)

In piercing time item:

Thickness of Sheet (T_s)

In cutting time item:

Cut Length (L_{cut})

Thickness of Sheet (T_s)

2.8 Welded assembly process

In Assembly Time item:

Part Weight (W)

Total Length of joining lines per part (Lj)

2.9 Flame cutting profile process

In piercing time item:

Thickness of Sheet (Ts)

In cutting time:

Cut Length (Lcut)

Thickness of Sheet (Ts)

2.10 Drilling process

In machine time item:

Hole diameter (dm)

Length of the Hole (Vs)

By classifying this information, it follows that there are five types of geometrical information that need to be inputted to CeDeas:

Distance between two points/lines (includes length, width, height and the distance between two holes, etc.)

Diameter

Area (includes surface/projected area)

Volume

Angle

Some of these values can be obtained directly from the properties of the entities in AutoCAD, but some of them do not have corresponding properties. For example, the distance between two holes in general will not have a corresponding entity (such as a line) to indicate the distance. For such values, the user will have to select points that can be used to compute the geometrical property.

Table 2.4 gives the list of the geometric entities in an AutoCAD drawing that CeDeas may require from. It lists the different cases of the geometric information relating to the manufacturing information and the methods that can be used to get the values. The "Need user define" column indicates that the user can select objects or define them manually. If the user can directly select the entity in the AutoCAD drawing, then the handle of that entity can be saved in the CeDeas database. For example, the length of a line can be the part width, height, thickness, etc.

Geometric information required by CeDeas	**Different cases of information representation**	**Different methods to get manufacture information**	**Need user define**
width, height, thickness, distance	Length	use StartPoint and Endpoint to calculate	no
	component that forms 3D Solid (such as a line that forms the edge of a box)	user defines start point and end point	yes
	distance without line connection	user defines start point and end point	yes
hole diameter	Circle	radius property of circle	no
	two parallel lines	user defines these two lines	no
	component of 3D solid	user defines start point and end point	yes
	component of standard entity in AutoCAD library (such as standard hole)	user defines start point and end point	yes
blended angle	two lines	user defines these two lines that form this angle	no
	two sections of polyline	using the control points of the polyline	no
the projected area of regular surface (such as circle, rectangle, arc)	area of circle and arc	area property of circle and arc	no
	area of triangle	area property of the polyline that forms the triangle	no
	area of region	area property of region	no
the projected area of non-regular surface	Area non-regular surface	area property of spline that forms the non-regular surface	yes
Volume	volume of 3D solid	volume property of 3D solid	no

Table 2.4 Geometrical information in AutoCAD drawing required by CeDeas

2.7.4 The Advantages of Linking CeDeas to AutoCAD

Linking CeDeas to AutoCAD can give the designer two advantages: it reduces the calculation time and improves accuracy. This will be illustrated in the following section.

2.7.4.1 Reduce Calculation Time

Although several CAD representation model schemes, such as B-Rep and CSG (Constructive Solid Geometry), are available in AutoCAD, applications that can help a user to obtain manufacturing information directly from AutoCAD drawing are not available. The data in STEP, IGES and DXF that are used for exchanging geometric information between different CAD applications are just arrays of values, and much research work has been done on feature recognition by these schemes. However, without background knowledge of such exchange schemes, one would not be able to understand what these schemes are representing.

In an AutoCAD drawing, the user can use automatic dimensioning routines to generate dimensions to indicate the geometric information that he/she is concerned with. This is quite suitable in the final design stage to help the user to obtain the manufacturing information from these dimensions, but not in the conceptual design stage. At the conceptual design stage, as shown in Fig 2.2, the designer has more freedom to change the scheme of the product he/she is planning than at any other stage. Modifications in the early stages also occur more frequently than that in later stages. If the product is complex, it will require a substantial amount of time to modify the dimensions in addition to the geometry. Further, the dimensional texts and lines in the drawing will clutter the drawing and hinder the development of alternative ideas. Another reason not to use dimensions for conveying information to CeDeas, is that most CAD systems now make a strict distinction between creating the CAD module and creating the dimensional views.

On the other hand, there are some types of information, such as the volume of a part or the area of a surface, that are implicitly embodied in an AutoCAD drawing. The designer first needs to find the particular geometric components that form this feature, and then get this information from the drawing after calculation. It will also cost time to obtain this geometry. The link module can help the user to search directly in an

AutoCAD drawing and to then use this information for specific analysis applications quickly and easily.

2.7.4.2 Reduced Susceptibility due to Human Error

Manual calculation and re-calculation of the geometric parameters, such as the area and volume of complex parts and surfaces, is a time-consuming task. The designer often divides the complex shape of a geometric entity into several simple shapes, such as a block, prism, wedge, pyramid, sphere, cone, cylinder and so on, by using the three Boolean operations, i.e. the union, the intersection and subtraction. These calculations also need a certain degree of background knowledge and experience. It is not possible to completely prevent human error in these calculations. Since the link module can obtain the manufacturing information directly from an AutoCAD drawing without the need for the user to input the values, it will improve the accuracy of the process.

CHAPTER 3 LINK MODULE IMPLEMENTATION

In this chapter, the programming language used for developing the link module will first be introduced. Then the author will look at the hierarchy and structure of the link module. This is followed by a discussion of the database structures in CeDeas and the link module. The main functions of the link module will finally be given.

3.1 PROGRAMMING LANGUAGE

In order to interface with both AutoCAD and CeDeas (which was written in Delphi), Borland C++Builder was chosen as the programming environment for the link module. Delphi was used to make some essential changes in CeDeas and COM technology is used to interface the link module to AutoCAD.

3.1.1 Borland C++ Builder

Borland C++Builder is an object-oriented, visual programming environment for the rapid application development of Windows applications. Borland C++Builder provides a comprehensive approach, using VCL (Visual Component Library) and RAD (Rapid Application Development) for flexible programming.

The programming language used for the link module is C++. It is an extension of C, which is the preferred development tool for the MSDOS, OS/2, Microsoft Windows, Microsoft Windows NT and UNIX operating systems [Linthicum and Klein, 1994]. C++ is one of several program languages that use a style called object-oriented programming. Using this style assists programmers in writing large programs that are correct, readable, modifiable, affordable and efficient.

Traditional programming languages, including FORTRAN and C, force the user to communicate with the computer in a demandingly simplistic manner. C++ and an object-oriented programming style elevate the communication to a more abstract level: they provide a means for investing intellectual effort to produce better quality programs and thus better quality science and engineering, for a given programming project [Lippman, 2000].

The object-oriented programming style mainly uses objects or classes to structure the program. An object is a set of information that consists of properties and functions used to manipulate the properties, and it may be collected into an interface such as a DLL (Dynamic link library). In general, it is a piece of binary software that performs a specific programming task.

An object (also called a class in C++) is a collection of data members and functions that work together to accomplish a specific programming task. In general, each class type represents a unique set of objects and the operations (methods) and conversions available to create, manipulate and destroy such objects [Borland C++ on-line help].

In terms of objects, C++ can provide the following important benefits [Barton and Nackman, 1994]:

> Direct concept expression: Objects are natural metaphors for both physical objects and abstract entities. Expressing computations in terms of objects reduces the gap between concept and program.
>
> Development malleability: Good programs evolve; evolution is easiest when the modifications are local. Objects combine data with functions to manipulate that data—allowing localization—and restricting access to an object's data —enforcing localization.
>
> Function extensibility: Using inheritance, new objects and their behaviours can be defined as incremental modifications and extensions of existing objects.
>
> Class abstraction: Using polymorphism, similarities among objects can be expressed in the program, allowing the programmer to write code in terms of the similarities without regard to the differences.

As a development platform, Borland C++Builder provides the full support for C++ and extends the base classes of C++ significantly. By using these extensive Borland C++Builder classes, a programmer can easily deploy general-purpose applications, international applications, database applications and COM-based applications.

3.1.2 Delphi

Delphi is similar to Borland C++Builder and Delphi was also developed by Borland Corporation. It is an object-oriented, visual-programming environment and has the same VCL and RAD capabilities as Borland C++Builder. Many of the traditional requirements of programming for Windows can be handled by a programmer within the Delphi class library, and Delphi can be used for the efficient development of a complicated or even a merely repetitive application.

Using Delphi, a programmer can create efficient Microsoft Windows and Windows NT applications. The key features in Delphi include:

An integrated development environment

A comprehensive library of reusable components

Integrated database connectivity

Internet/intranet development support

ActiveX support

3.1.3 Comparison Between C++ Builder and Delphi

The biggest difference between Borland C++Builder and Delphi is that Delphi is based on the Object Pascal programming language instead of on C++. Object Pascal is the latest rendition of the highly successful Turbo Pascal language that Borland originally introduced in the 1980s [Barton and Nackman, 1994].

Pascal, which is the programming language used in Delphi, has a different programming syntax than that of Borland C++. Although Delphi has the same easy programming advantages as C++Builder, it is inflexible for many programmers who have the knowledge background of C and C++. Such inflexibility will cause the development and maintenance of CeDeas to be difficult, since C and C++ has been used to develop most popular MSDOS and Windows program such as Paradox, Microsoft Word and Excel [Xiao et al., 1999].

Moreover, C++Builder has a good compatibility with Delphi; in other words, C++Builder can compile Delphi code with little or no modification. In order to obtain a flexible way to develop the link module relating AutoCAD and CeDeas (which was written in Delphi), Borland C++Builder was chosen as the development environmen.

Using C++ to program Windows applications is also more popular than using Pascal. Until Delphi was released, the Pascal language was nearly extinct. Despite this, C has been, and probably always will be, more popular than Pascal [Friedman and Koffman, 1994].

3.2 THE HIERARCHY AND STRUCTURE OF THE LINK MODULE

One natural limitation of a designer's cognitive abilities is his/her limited knowledge of the details of a wide variety of manufacturing process constraints and cost, as well as the shortage of getting on-line and accurate manufacturing information from an AutoCAD drawing. Such limitations lead to the possibility of introducing high or inaccurate component costs into the detail design. The link module developed here attempts to link AutoCAD and CeDeas in order to get the manufacturing information directly from an AutoCAD drawing, and to then transmit it into CeDeas. The manufacturing cost can then easily and quickly be obtained in the latter conceptual design phase or in the early detail design phase. The structure of the link module is illustrated in Figure 3.1.

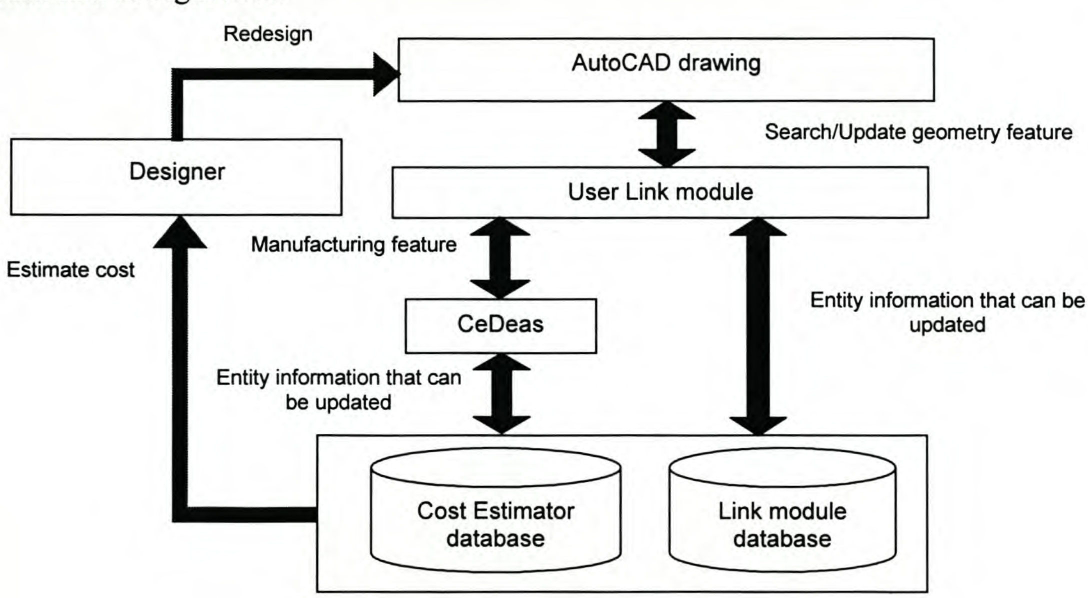


Figure 3.1 Link module structure

After the conceptual design, the designer would typically have rough drawings of the parts and the assembly of a product (e.g. in the form of initial layout drawings). In order to arrive at a low cost product, the designer can use CeDeas to obtain a rough estimate of the cost of that product and compare different manufacturing options to optimise the design.

When using CeDeas, the designer starts CeDeas when creating a project, and then subdivides the project into assemblies and parts. As mentioned in Chapter 4, manufacturing information, such as the length of a part, the area of a part surface, etc., must be inputted into CeDeas as parameters for each part and assembly to calculate the cost of that product. The link module then will prompt the designer to directly select the appropriate entities in an AutoCAD drawing, transmit the required manufacturing information to CeDeas and also save the pertinent information in a database for the purposes of future updating. Figure 3.2 shows a flow chart of the procedure followed by the link module.

Using these values and other non-geometric information, such as the machine power, set-up method, etc., a designer can obtain the estimated cost of the product. If the result is acceptable, the designer can start to implement the detail design of each part and assembly. If the result of the cost calculation is not acceptable, the designer must return to the original design to make adjustments. CeDeas can give the cost contributions of different parts/assemblies, from which the designer can get an overview of the product cost and optimise the design. The data saved in the database can be regarded as manufacturing feature class instances. After the designer has adjusted the design, and unless the appropriate entities in the AutoCAD drawing were deleted, the link module can get the new values of the changed entities without requesting the user to select them again. If an entity was deleted, the link module will ask the designer to redefine it. After a new parameter value has been determined, the link module can overwrite the old value and save it in the database. By using this data, CeDeas can get the latest estimate of the cost to assist the designer in the detail design decision-making on the design.

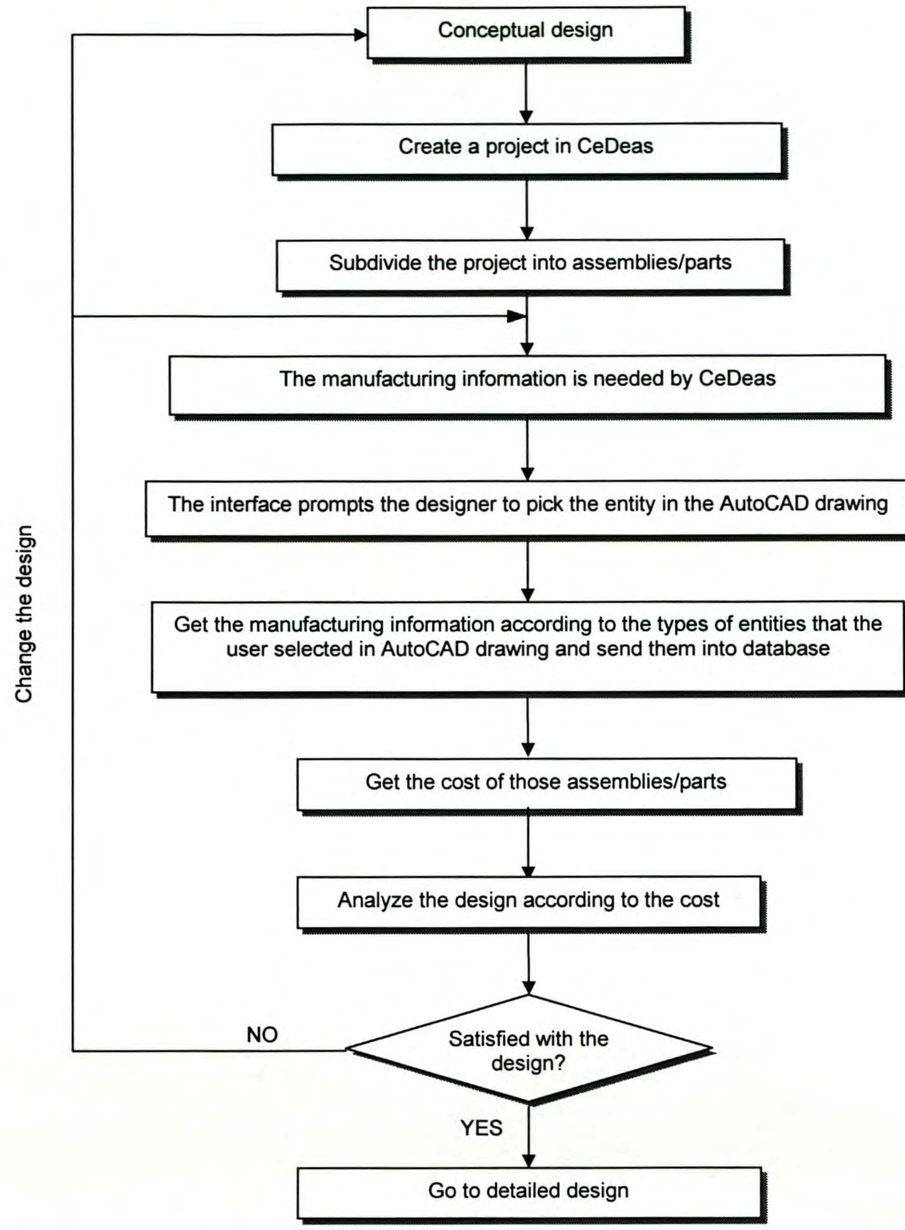


Figure 3.2 The chat diagram of procedure of the interface hierarchy

3.3 THE STRUCTURE OF THE DATABASES

Both CeDeas and the link module use databases to enable the programmer to record, modify and access the user input information. An entity in a database is an object for collecting the input information from the user. A group of related entities becomes an entity set or table. The table is perceived as a two-dimensional structure in which each row defines a single entity within the entity set, and each column defines a particular attribute. The content of the tables in CeDeas databases and the link module database are discussed in the following sections.

3.3.1 The Database of CeDeas

In order to manage the project, various entity sets, e.g. headings, part details, process plans, element times, etc., need to be recorded. The link relationships between these entities must also be recorded so that different projects can share the same database structures. The relationships between the databases that affect the link module are shown in Figure 3.3 [Schuster, 1997].

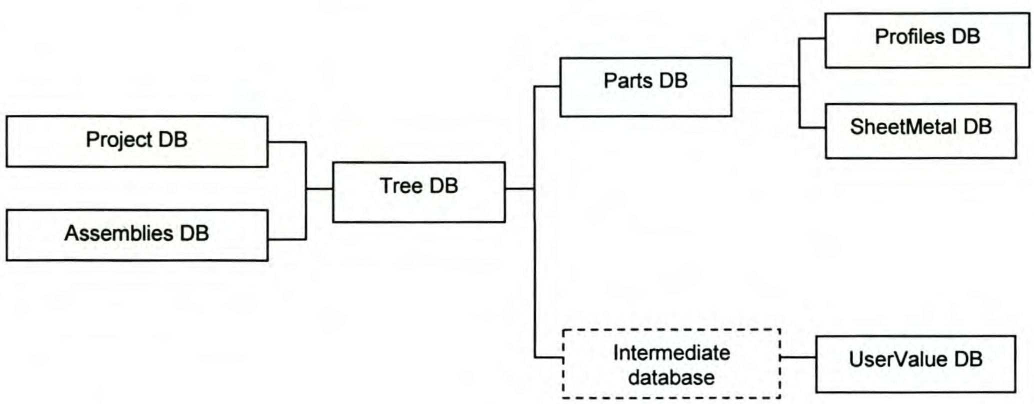


Figure 3.3 Relationships between databases interacting with the link module [Schuster, 1997]

The CeDeas database is handled by Paradox [1995] in Delphi. It is an SQL-based relational database and includes all the input data, calculation references (such as the equations, constants etc.), calculated results, memos, assemblies, parts, etc. Each row in a table has a primary key, which can be used to identify each row uniquely. The database exhibits both data independence and structure independence. By using the ID of each database, the databases can share common attributes that enable one to link them together. Table 3.1 give a description of these databases (ID number is used to distinguish different entities).

In order to link CeDeas to AutoCAD, a handle field was added to the parts database. In the profiles database in CeDeas, the length handle field and the volume handle field were added. In the uservalue database; the length, width, thickness and projected area handle were added to the sheet metal database to store the different handles that come from an AutoCAD drawing. Since some input entities do not need to be linked to AutoCAD, the default value is set to "NoHandle" to indicate that the value of the entity does not come from AutoCAD. If the user selects entities in an AutoCAD drawing, the send database function can overwrite the default value and save the handle in this field. By using the handle, the link module can retrieve the AutoCAD drawing in order to update these values after the user has changed the design.

Database name	**Description**
Project	Store the project name, description, work date and the use status.
Assemblies	Store assembly drawing number, name and description.
Tree	Store the information of the tree entity that is shown on the main window.
Parts	Store the part name, drawing number, description and the link information.
Profiles	Store profile ID number, part length, part value and section ID that is used to link data in other database.
SheetMetal	Store sheet metal ID number, part length, part width, projected area and sheet ID that is used to link data to other databases.
UserValue	Store the valuelink ID number, parameter ID and user value.

Table 3.1 Description of main databases related to the link module

There are other databases that are used to save the calculation equations, library items, material constants and link information that connect different databases, but they are not connected to the link module. Appendix A gives a detailed description of these databases.

3.3.2 The Database of The Link Module

The database of the link module is used for storing the detailed information of the entities that the user selected in an AutoCAD drawing. Table 3.2 describes the information that is contained in the link module database. It includes the handle, types, memo and the values that were required by CeDeas.

Field name	Description
Name	Stores the TypeName of selected entity in AutoCAD drawing
Fname	Serves as an index to get the corresponding geometrical value, such as L stands for the length, A stands for area. Store the single letter to identify which method can be used to calculate the required value.
Value	Stores the single value of one entity or each value of a composite entity.
FinalValue	Stores the data that is transmitted to CeDeas.
MasterID	Indicates whether it is a single entity or composite entity, 0 — single, 1 — composite.
SelectType	Identifies the select method of a selected entity, 0 — direct select, 1 — manual select.
Handle	Stores the handle of the selected entities.
DrawingNo	Stores the drawing number that the selected entity belongs to.
CEDescription	Stores the project, assembly, part and process that the selected entity belongs to.

Table 3.2 Description of the link module database

Similarly to CeDeas, the link module database is managed by the SQL-based relational database Paradox, which is accessed through the BDE (Borland Database Engine). It is easy for a programmer to design, manage and access a database as a logical database without having to be concerned about the details of physical storage, access paths and data structure.

A relational database exhibits both data independence and structural independence. Within a table, each row has to define a single entity. Because the handle in an AutoCAD drawing is unique, the link module database uses the handle as the primary key (index ID field) to distinguish between different entities in the link module database. However, because one entity could be stored in multiple cases, the handle cannot be unique as an index field. The record number must be added to the handle to make it unique, using the format record number+%+handle (the symbol "%" is used for the program to distinguish the record number and the handle). For example (see Figure 3.4), the length of line A can be considered as the width or the edge length used to calculate the area of a machined surface. Then the handle of line A needs to be saved in the link module database twice. By setting the handle of line A to be saved twice without causing conflicts in the primary key and with the *MasterID*, the program can easily identify whether the saved value is the value of a single entity or is the value of a composite entity that consisting of several single entities.

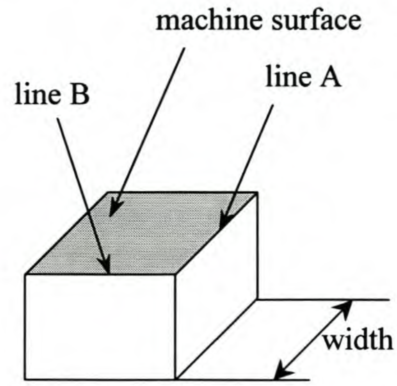


Figure 3.4 A machine part example

3.3.3 BDE Setup

A BDE alias is a collection of configuration parameters used to organize the connection between the BDE and a given database. Aliases are usually database specific and based on BDE database drivers. In order to set up the link module database in different computers, the program has a function named *ChangePath*. At the start of this program, it automatically checks whether the database alias exists. If it does not exist, the program uses this function to set up the database configuration. Once the alias of the link module database exists, the *ChangePath* function will not be called up at the beginning of the manipulation of the program.

3.4 MODIFICATIONS IN CEDEAS

In order to connect the link module in Borland C++ Builder directly to CeDeas, two buttons named *To AutoCAD* and *Update* were added to the user interface to indicate the *select mode* and *update mode* respectively. Adding such buttons can only be done in the Delphi development environment. After adding all the Delphi units into the link module and building all the Delphi code into the C++Builder environment to form a project, both the CeDeas code and the link module code are components of the same project in the C++ Builder environment. However, the changes made in the Delphi code have to follow Delphi syntax.

The buttons named *To AutoCAD* and *Update,* when implemented in CeDeas Delphi code, behave like triggers that activate the user interfaces of *Select mode* and update mode respectively. This was done by defining the pointer of these two forms first, then using the show function in C++ Builder to energize these two windows, from which the user can employ the functions in the link module.

Another change that had to be made in Delphi was to modify the functions that write to the database in CeDeas. Since the handle field in the relational database had been added, a correspondingly changed SQL (Structured Query Language) code was needed. The Delphi code was changed by adding the field name in the database into the operation, so that when CeDeas saves the data into the database, the other information can also be saved into the database.

3.5 MAIN FUNCTIONS IN THE LINK MODULE

In order to get the geometrical information from an AutoCAD drawing and to convert it into manufacturing information, two modes are applied in the link module in the phase between the conceptual design and the detail design. These are *Select Mode* and *Update Mode*, which will be described in the following sections.

3.5.1 Select Mode

Select mode is used to get geometric information from an AutoCAD drawing and then to convert it into the manufacturing information that CeDeas needs to calculate the cost of a certain part or assembly. Figure 3.5 shows the user interface of the *Select mode* in the link module. The user triggers an event (e.g. pressing the *To AutoCAD* button) on the input dialog box in CeDeas to launch the *Select mode*.

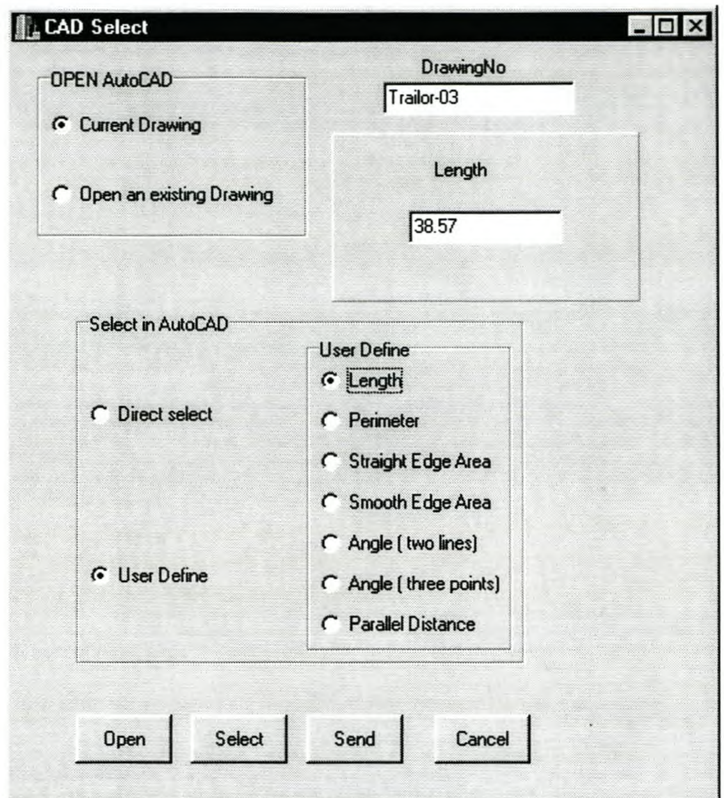


Figure 3.5 User interface of select mode in the link module

Select mode has the following capabilities:

- Gets the AutoCAD Application automation as base object.
- According to the different requirements in CeDeas, shows one of the select options from the user value input, profile and sheet metal input options.
- Prompts the user to pick entities in the AutoCAD drawing that are related to CeDeas.
- Converts the geometrical information from the AutoCAD drawing into the manufacturing information that is needed by CeDeas.
- Prompts the user to define the manufacturing information manually in the AutoCAD drawing.
- Saves the manufacturing information of the entity that the user has selected into the database of the link module.

3.5.1.1 Open AutoCAD

There are two options for connecting to an AutoCAD drawing when clicking on the *Open* button:

- Current Drawing: This is the default setting in this item. This option can be chosen if the user is currently working in the drawing that relates to CeDeas's current part or assembly. With this option, a user can click the Select button directly to select the graphic entities in AutoCAD drawing.

- Opening Exist Drawing. This option can be chosen if there is no AutoCAD application running or the user is not working on the AutoCAD drawing that is required by CeDeas. With this option, an AutoCAD application will be executed or a dialog box will show up to prompt the user to open the appropriate drawing.

The following C++ Builder codes briefly illustrates this Open function:

```
Variant CADObject,CADDoc;
      //Current Drawing option is checked
if(RadioGroup1->ItemIndex==0)
   { CADDrawing=GetActiveOleObject("AutoCAD.Application");
      //open Exist Drawing option is checked
elseif(RadioGroup1->ItemIndex==1)
   {CADObject = CreateOleObject("AutoCAD.Application");
    CADDoc = CADObject.OlePropertyGet("ActiveDocument");
    CADDrawing = CADDoc.OleFunction("open");}
```

Variant is a type of variable in Borland C++Builder that is capable of representing values that can change type dynamically. Whereas a variable of any other types is statically bound to that type, a variable of the Variant type can assume values of differing types at run-time. This is suitable for the situations where the actual type to be operated upon is changeable or unknown at compile-time. Since variants can contain OLE Automation objects, it can be used to refer to the AutoCAD automation object in the link module. The programmer can call the object's methods and read or write to its properties through the variant. However, it needs to be noted that although

Variant offers great flexibility and is suitable for representing OLE objects, it also consumes more memory than a regular variable, and operations on the variant are substantially slower than operations on statically typed variables.

Borland C++ Builder provides OLE functions that can perform the operations on OLE objects. The *GetActiveOleObject* function is used to access the OLE running object table to get the object with the function parameter name that was specified. *CreateOleObject* creates a single un-initialised object of the class specified by the ClassName parameter of the function. *OlePropertyGet* returns the value of a property of an Automation object whose name would be the function parameter, and *OleFunction* executes an OLE procedure, function or a property Get or Set method. The *OleFunction* can use both named and positional parameters, which can be integer, real, Boolean, string and variant types. *AutoCADApplication* is the root object in AutoCAD. It has the ActiveDocument property that can be used to get the drawing file (named ActiveDocument in AutoCAD automation) as mentioned in Chapter 4.

If a user accidentally picks the entity in an AutoCAD drawing before getting the AutoCAD automation, the program would access an address that does not exist in the memory and cause unexpected problems. In order to prevent this situation, the "Select" button is only available after the AutoCAD Automation object has been found.

3.5.1.2 Drawing No

The title or serial number of the AutoCAD drawing that is related to CeDeas is shown in this item. The drawing number is the same as that shown in the project/assembly/part link module interface and can be modified by the user if it is not correct according to the arrangement of the project drawings. Note that this item is used as a reference to specify the AutoCAD drawing if the user gets confused between several drawing numbers.

3.5.1.3 Result Show panel

This item is used to display the selected results in a value edit box. It will show a different result panel depending on the different requests from CeDeas. Table 3.3 shows the different items that will appear according to the parent window.

Parent Window	**Display item**
User edit value input	User selected entity data.
Profile information input	Part length, part volume.
Sheet Metal information input	Sheet metal length, width and projected area.

Table 3.3 Display items in result panel

3.5.1.4 Selecting in AutoCAD

The link module provides the flexibility that enables the user to indicate a manufacturing entity in an AutoCAD drawing in two ways: direct select and user defined method. The units of the results are based on the drawing unit used in AutoCAD. Since CeDeas requires that the units of lines, perimeters, projected areas, volumes and angles are millimetres, square millimetres, cubic millimetres and degrees respectively, the user must set the metric units for the AutoCAD drawing. From the input dialogue box, there are two sets of input: standard input, such as the input of fixed length or volume, and the user-defined value, such as when the input changes according to a different part/assembly.

Direct select method

The user can directly pick an entity in the AutoCAD drawing if the Direct Select option is checked. This entity must have a property corresponding to the manufacturing information required by CeDeas. The link module can then access the manufacturing information related to the entity that the user picked, and show the results on the user interface of the link module. If the selected entity does not contain the required information, the link module will turn off *Select mode* and prompt the user to define the value manually. Figure 3.6 shows the flow chart of this mode.

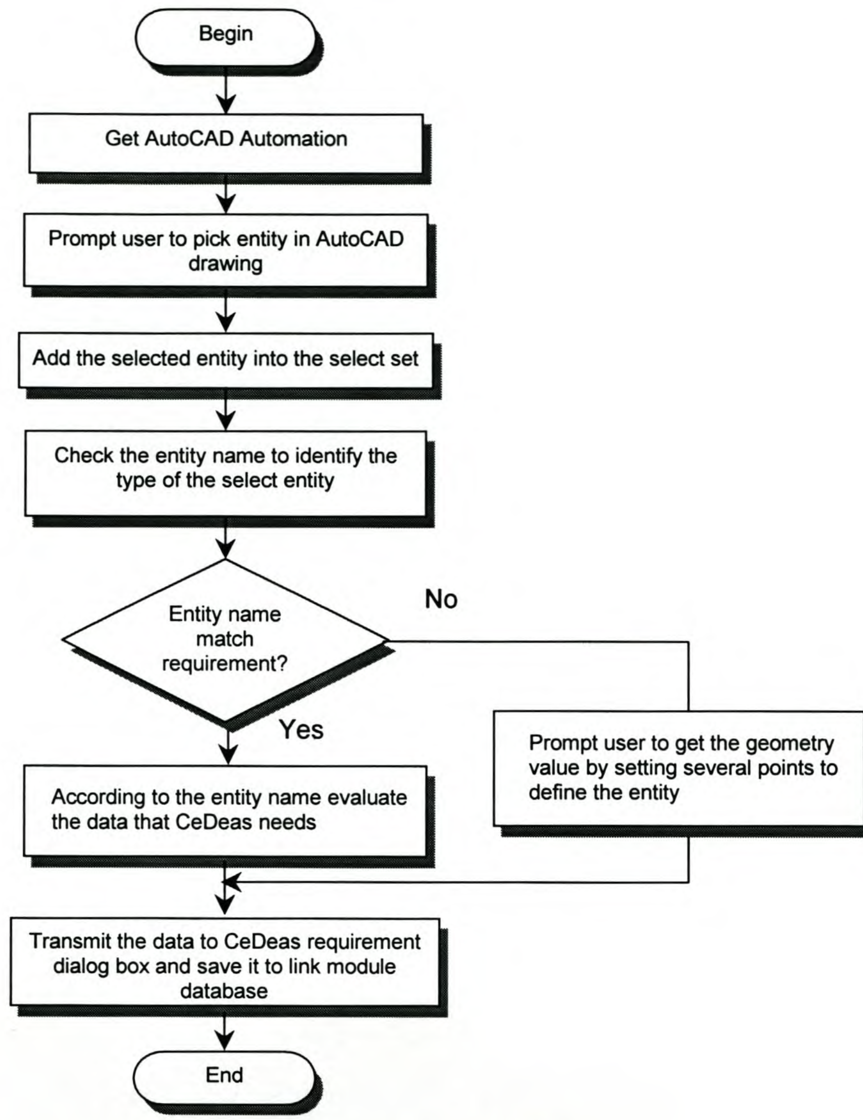


Figure 3.6 Select mode flow chart

To enable a user to directly pick the entities in an AutoCAD drawing, the link module must first create an entity collection to hold the entities that the user will select. The entity that the user has selected can then be added into this collection. The following code illustrates this function:

```
Variant CADSet,CADSelect;
CADSet = CADDrawing.OlePropertyGet("SelectionSets");
CADSelect = CADSet.OleFunction("Add","PickSelect");
        //CADSelect as the container of the Select object
CADSelect.OleFunction("Clear");
CADSelect.OleFunction("SelectOnScreen");
```

CADSet is the collection of all selection sets in the current drawing and *CADSelect* is a group of one or more AutoCAD objects specified for processing as a single unit. The procedure that the user follows to select entities in the AutoCAD drawing is similar to that used in the Select command in AutoCAD. There are several methods by which the *SelectionSets* object can get the selection; they are *Select*, *SelectAtPoint*, *SelectByPolygon* and *SelectOnScreen*. The *SelectOnScreen* method can be manipulated to provide the way in which to pick the graphic entity in AutoCAD drawing, through which the user can select more than one entity in the AutoCAD drawing on screen. In order to prevent confusion between different selected entities the *CADselect* must be emptied using the *Clear* method, each time the user selects a new entity. Once the user has selected the entities in the AutoCAD drawing, the link module can access the entities via variants.

In order to get the required manufacturing information from the geometrical information, the link module must first identify the type of geometric entity that the user has selected, because different geometric entities contain different potential manufacturing information (for example, a line in AutoCAD may indicate length, width or thickness, but not volume). In an AutoCAD drawing, each entity has its own properties, such as the Handle and Entity Name. The entity name is equivalent to the class name of the object. It can be used to distinguish between different classes in an AutoCAD drawing. There are 35 possible classes in AutoCAD R14 ModeSpace (refer to Figure 2.22 and see Appendix B for more detail).

As mentioned in Chapter 4, CeDeas only requires values for length, perimeter, diameter, area, volume and angle. Therefore only the entity types that have properties related to these parameters need to be considered in the link module. These entities types are listed in Table 3.4. AutoCAD uses different entity types to indicate different classes. The entity type is an integer that can be used in a C++ Builder *case* construct to distinguish between the different geometric classes. The *Value* entity in the Evaluating Function column in Table 3.4 means that the user can get the values of the corresponding manufacturing information directly via the properties of that entity type. The other evaluation functions are illustrated in the following paragraphs

Entity	Entity Type	Available Geometric Information	Properties/Method	Evaluating Function
3D face	1	Projected area	GetBoundingBox	GetBoxArea
3D polyline	2	Projected area	GetBoundingBox	GetBoxArea
3D Solid	3	Projected area	GetBoundingBox	GetBoxArea
		Volume	——	Value
Arc	4	Radius	——	Value
		Projected area	Area	Value
Circle	8	Radius	——	Value
		Projected area	Area	Value
Line	19	Length	StartPoint	GetLength
			EndPoint	
Polyline	22	Length	——	Value
		Projected area	GetBoundingBox	GetBoxArea
SplyLineLight	23	Projected area	Area	Value
Region	28	Perimeter	——	Value
		Projected area	Area	Value
Solid	30	Projected area	Coordinate	GetBoxArea
Splight	31	Projected area	Area	Value

Table 3.4 Entities in AutoCAD responding to CeDeas

GetBoxArea

GetBoundingBox is a method that is contained in all the drawing objects in AutoCAD. It can get the *MinPoint* (lowest point) and the *MaxPoint* (highest point) of a box enclosing the specified object. The corners are returned in WCS coordinates with the box edges parallel to the WCS X, Y and Z axes, these coordinates can be used for calculating the projected area. GetBoxArea uses the coordinates to calculate the projected area of a rectangle is that perpendicular to one of the X, Y or Z axis. Figure 3.7 shows a boundary box of a rectangle shaped part and its MaxPoint and MinPoint on the x-y plane.

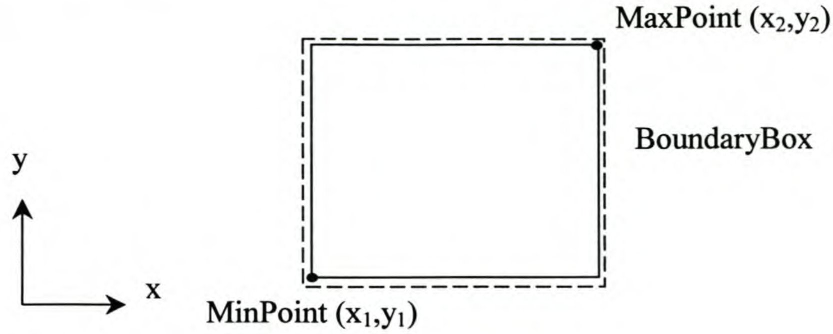


Figure 3.7 GetBoundingBox Method

GetBoxArea uses the following formula to get the projected area of the rectangle on x-y plane:

$$Planar\ Area = |(x_1 - x_2) \cdot (y_1 - y_2)|$$

This approach is also applied on x-z and y-z planes. The program prompts the user to identify which axis the rectangle is parallel to.

GetLength

The property list in the AutoCAD Automation documentation includes that a line object does not have a length property. Nevertheless, it has properties such as "StartPoint" and "EndPoint", which can be used to calculate the length of the line by applying this formula in the *GetLength* function (these two properties are expressed in the WCS coordinate in the AutoCAD drawing).

$$Length = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Note that the manufacturing information may be contained in several geometric objects. For example, Figure 3.8 shows that the length of a part is comprised by lines A, B and C. The part length's value is the sum of the lengths of A, B and C. No line directly indicates the part length. If a user picks line A, B and C, there can be two possibilities: the user either wants the length of the part or the user picks them in the small scale drawing without noting that additional lines were selected. For the length of a part, the link module will calculate the length value by adding the segment lengths together and saving the length value, handles of these three lines are saved into a vector and a string list, and show the part length in the user interface of the link module. If the user had selected one or more lines by mistake, the link module just clears the selection set and returns to the user interface.

If the graphic entity has more than one property relating to the required manufacturing information, the *Select mode* will allow the user to choose the correct property in the AutoCAD command line. For example, if the surface of a part is formed by a polyline, it has two properties: area and perimeter, and it will query the user about which property he/she needs.

Select mode uses A to represent area and P for perimeter, so that the user can decide which option to input into CeDeas.

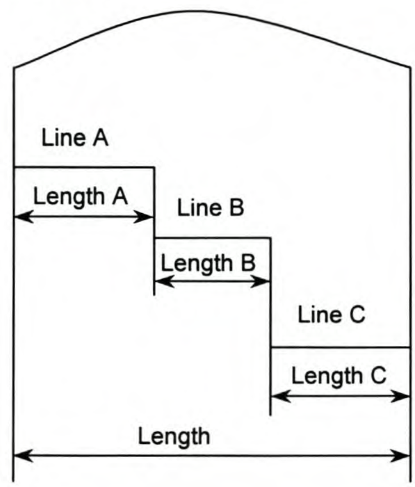


Figure 3.8 A multiple length example of a part

User define method

The user define method provides eight options that the user can use to define the entity in an AutoCAD drawing: Length, Perimeter, Straight Edge Area, Smooth Edge Area, Angle (two lines), Angle (three points), Parallel Distance and Length. They are described in the following paragraphs.

<u>Length</u>

The user can pick two points to get the distance between them by using the GetLength function. It suits the following cases:

- The length of a line that is a component of a block (such as an edge line of a box). No handle and property are available for this line, since only the handle of this block exists in the AutoCAD drawing, unless the block is exploded.

- The distance between two points without a connecting line (such as a gap or a V-groove distance).

- The length of a segment of a line. For example, in Figure 3.9 it is required to get the diameter that is indicted by points A and B, which belong to line L_1. The start point and the end point are not A and B and it is difficult to identify A or B as the start point or the end point of L_2 and L_3.

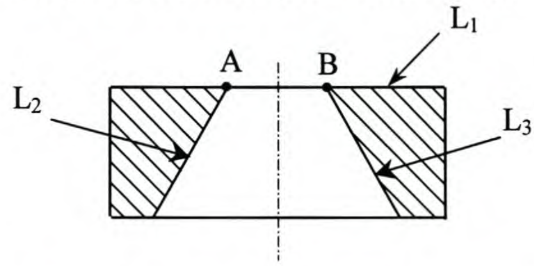


Figure 3.9 Example of length evaluation

Since the link module only receives the WCS coordinates of the two points that the user picks in the AutoCAD drawing, no new entity is defined and no *Handle* property is available. The link module can only save the value of the calculated length in the database without the *Handle* property.

<u>Perimeter</u>

With this option, a user can specify a list of points to define several vertices according to the surface shape of which the perimeter needs to be evaluated, and then get the sum of the lengths of these segments.

<u>Straight Edge Area</u>

The straight edge shape of a part would be a triangle, parallelogram, trapezoid, regular polygon or irregular polygon. Although there are formulas that can be used to get the area of the projected surface by breaking it into regular geometric shapes, it would be time consuming and the accuracy is limited if it is a complex shape.

Note that the Area property in the polyline object in AutoCAD can give the area enclosed by that polyline. The link module uses the approach to define a polyline that has the same profile as the projected region to get the required area. Firstly, the user must pick a sequence of vertices of the projected region. The link module can generate a polyline that has the same shape as the projected region by using

AddPolyline method in *ModelSpace*. Using the *Area* property of that polyline, it then is easy to get the value of the area of the projected region. If the polyline is not closed, the area is computed as if a straight line connects the starting point and the endpoint as shown in Figure 3.10. After getting the projected area, the link module deletes this polyline using the *Erase* method belonging to the polyline object. If the area is not aligned with the X-Y plane, it needs the user to define the vertices of the projected area on the X-Y plane. Using such vertices, the user can get the projected area.

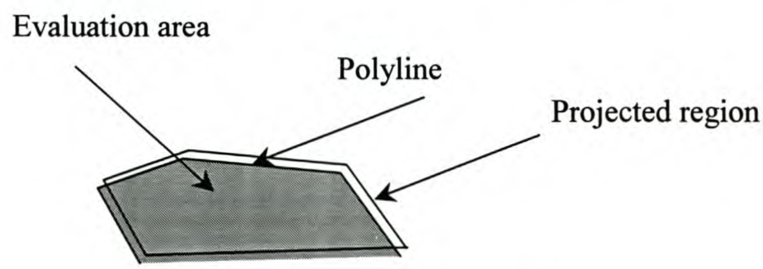


Figure 3.10 Evaluate the area using polyline

Smooth Edge Area

For a part with a sculptured surface that is not created by a spline, or a composite surface that is constituted by arcs and lines, using a polyline to represent the projected region's edges is far less accurate than using a spline or calculating it manually to evaluate the area of a part.

A NURBS spline as mentioned in Chapter 3, can provide the necessary flexibility. It is a smooth curve passing through a given set of points that lie on the desired path of the curve. *Addspline* in *ModelSpace* can be used to generate a spline in an AutoCAD drawing. The user needs to define a list of control points, the start and end tangent points, to follow the part surface. The spline must be a closed curve. Note that the area also can be computed as though straight lines connected the starting point and endpoint, as in a polyline, but this will give the incorrect result. After retrieving the area property of the spline, the link module calls up the spline object's *Erase* method to delete the spline defined by the user.

Angle (two lines)

The manufacturing information required by CeDeas includes angle, but there is no angle object available in AutoCAD Automation. This option is suitable for evaluating an angle that is formed by two lines. The angle is determined by evaluating the different slopes of the equation of these two lines.

The general equations of two selected lines on the homogeneous planar can be:

Line A with start point (x_{11} y_{11}) and end point (x_{12} y_{12}):

$$\frac{y-y_{11}}{x-x_{11}}=\frac{y_{12}-y_{11}}{x_{12}-x_{11}} \qquad (x_{11} \neq x_{12})$$

Line B with start point (x_{21} y_{21}) and end point (x_{22} y_{22})

$$\frac{y-y_{21}}{x-x_{21}}=\frac{y_{22}-y_{21}}{x_{22}-x_{21}} \qquad (x_{21} \neq x_{22})$$

The gradients of these two lines are:

Line A: $m_A=\dfrac{y_{12}-y_{11}}{x_{12}-x_{11}} \quad x_{11} \neq x_{12}$

Line B: $m_B=\dfrac{y_{22}-y_{21}}{x_{22}-x_{21}} \quad x_{21} \neq x_{22}$

In the case where the line is vertical (when $x_{11}=x_{12}$ and $x_{21}=x_{22}$), the link module will directly set the result to 90 degrees.

The value of the angle between these lines A and B is:

$$artan(\frac{m_B-m_A}{1+m_A \cdot m_B})$$

In the case where $m_A \cdot m_B$ equals -1, the link module will also directly set the result to 90 degrees.

This procedure is performed by the function GetAngle in the link module. The calculation value will be between 0 and 90 degrees. The program asks the user to affirm whether it is an acute angle or an obtuse angle. The default return value is acute angle. The angle value and the Handle of these two lines can be saved in the database, and it can be updated in the future. See Appendix B for more details about *GetAngle*.

Angle (three points)

This option is suitable for the case where an angle is formed by a continuous line such as a polyline, which prevents the user from picking two lines to evaluate the angle. Nevertheless, an angle can be defined by specifying a start point, a vertex and an end point. Calling the GetAngle function will return the value of that angle and the calculation value is also between 0 and 90 degrees. Handles that specifiy the angle are not available, so the link module can only save the value and it has to be redefined when updating the cost estimation.

Parallel Distance

This option is suitable for calculating the distance between two parallel lines, for example as shown in Figure 3.11. The centre distance between holes A and B does not have a corresponding geometric entity and has to be obtained by specifying two points. The method here is to use the distance between a point and a line. This point is defined by the midpoint of the line that the user first selected,

The distance between point (x_0, y_0) and line $A \cdot x + B \cdot y + C = 0$ is [Math Lecture Group, 1988]:

$$\frac{|A \cdot x_0 + B \cdot y_0 + C|}{\sqrt{A^2 + B^2}}$$

The start point (x_1, y_1) and end point (x_2, y_2) that is obtained from the line's properties can be used to determine the coefficients of the equation:

$$A = \frac{y_2 - y_1}{x_2 - x_1},\ B = -1, C = y_1 - \frac{y_2 - y_1}{x_2 - x_1} \cdot x_1$$

For the case in which the line is vertical ($x_1 = x_2$), the distance is just $|x_1 - x_0|$.

This option is also suitable for getting the diameter of a hole, as in Figure 3.11. The value and Handle property of this option can be saved in the link module database and can be used for updating.

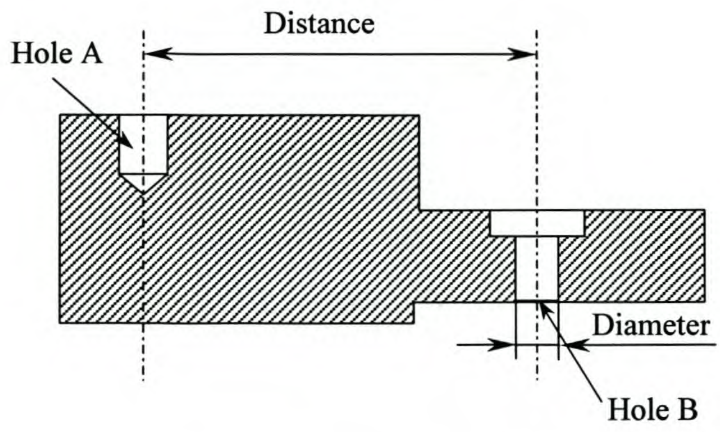


Figure 3.11 A case of parallel distance

3.5.1.5 Send

The results shown on the link module user interface and the handles of the entities that the user selected in the AutoCAD drawing can be sent to CeDeas by pressing the Send button. It uses the function SetDatabase to save them into the database in CeDeas.

The procedure for saving the information in Figure 3.11, for example, is performed by calling the SetDatabase function as follows:

```
void __stdcall SetDatabase(AnsiString DrawingNo,int PType,int
        Pm, double PFValue, AnsiString PName, AnsiString
        PFName,TStringList* PHandleList, double
        PValueList[10],AnsiString PCEName)
```

Each parameter of this function corresponds to the different database fields. Appendix C gives the detailed code of this function (the bolded text indicate the variable types in Borland C++).

3.5.2 Update Mode

After a cost estimate of a product has been done, the designer probably needs to change the design where the cost is not acceptable. To obtain the revised cost estimate of a changed product, the manufacturing information in CeDeas should be updated. The update mode is used for this propose, as well as to recalculate the manufacturing cost of the parts and assemblies. The user interface is shown in Figure 3.12. When the user interface appears, the update mode acquires the AutoCAD Automation interface as described in 3.5.1.1 to facilitate the later operations. There are several functions in the update mode that are activated by clicking the respective buttons. The functions associated with the buttons are outlined in the following paragraphs.

Cancel

It closes the user interface without doing anything.

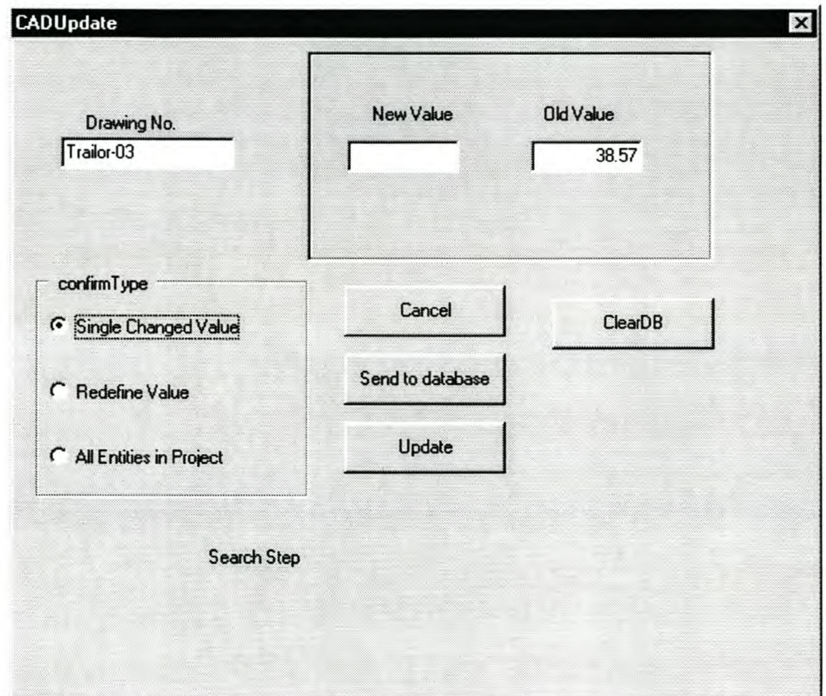


Figure 3.12 Update mode user interface

Send to database

It sends the updated values to the database and overwrites the old values. It also compares the old and new values, and if they are the same, the data will not be overwritten. The values include all items in Table 3.2 to save all the relating information.

Update

Since recalculating the cost estimation could have been preceded by changing the project tree, the user must descend the project tree to the changed entity. For each changed entity with a handle relating to it, the update mode will automatically get the new value according to the handle in the CeDeas database, unless it was deleted. For the entities that were defined manually, it will prompt the user to redefine it. The flow chart is shown in Figure 3.13. There are three options relating to this button: Single Changed Value, All Entities in Project and Redefine Value.

Single Changed Value

This is only for updating the single value that the designer is working with. In response to the request from CeDeas, the update mode gets the corresponding handle from the database in CeDeas. By using the handle, the link module gets the *drawing number*, *entity name*, *MasterID*, etc. from the link module database. If there is no *handle* available (the saved string in Handle field is "NoHandle"), the update mode will show the redefine user interface to allow the user to define the manufacturing entity. If the handle is available, the update mode adds all the AutoCAD drawing's graphic entities into the selection set by using code such as the following:

```
Variant CADUpDateApplication, CADUpDateDrawing, CADUpdateModel;
Variant CADUpdateUtility, CADUpdateSet, CADUpdateSelect;
CADUpDateApplication = GetActiveOleObject("AutoCAD.Application");
CADUpDateDrawing=CADUpDateApplication.OlePropertyGet("ActiveDocument");}

CADUpdateMdoel = CADUpDateDrawing.OlePropertyGet("ModelSpace");
CADUpdateUtility = CADUpDateDrawing.OlePropertyGet("Utility");
CADUpdateSet = CADUpDateDrawing.OlePropertyGet("SelectionSets");
CADUpdateSelect = CADUpdateSet.OleFunction("Add","CADupdate");
CADUpdateSelect.OleProcedure("Select",5);
    //Select all entities in AutoCAD drawing
```

The method that selects all the graphic entities in the AutoCAD drawing is one of the options of the select method, e.g. the selection set. A loop can be formed to search all the graphic entities in the AutoCAD drawing. Using the entity count in the selection set, the link module will get the *handle* property of each entity, and then compare each saved handle value from the CeDeas database. If the saved handle value in AutoCAD matches the entity name, the update mode will calculate the appropriate value.

In the case that the manufacturing entity is composed of several geometric entities, as indicated by the *MasterID* field in the link module database, the update mode gets several handles, searches each of them to get the updated values and adds the individual values together to obtain the required value.

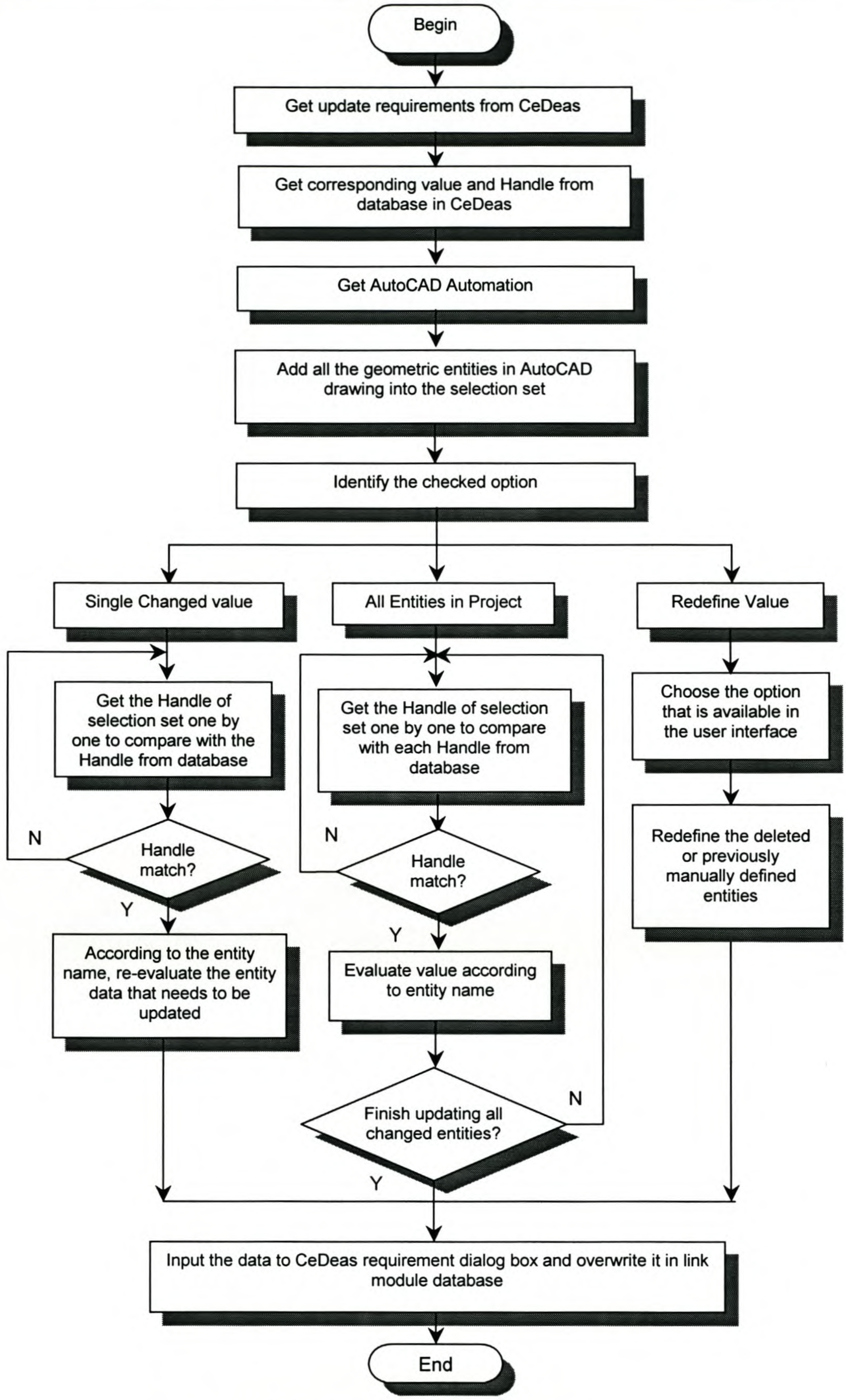


Figure 3.13 Flow chart of the update mode

All Entities in Project

This option is for updating all the changed values that have available handles. With this option, the update module gets all the handles of a project in the CeDeas database. For each handle the same procedure as for the Single Change Value option is followed. By applying a loop, the user can get all the changed values of the entities with *handle* properties available in an AutoCAD drawing. It takes time to compare each handle from the CeDeas database with all the entire entity handles in the AutoCAD drawing and to display the relevant information in the different text boxes on the screen.

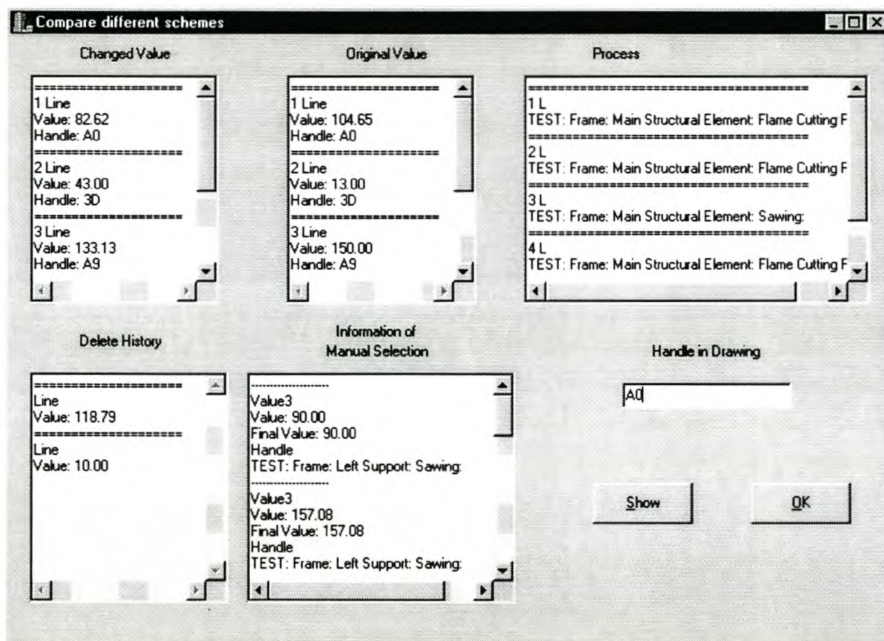


Figure 3.14 Update all values user interface

The result window is shown in Figure 3.14. It contains such information such as:

- Changed Value: This box lists each changed value, as well as the name and handle of the directly selected entity in the link module database.
- Original value: This box lists each original value, and the name and handle of the directly selected entity in the link module database.
- Process: This box lists the related process information of each directly selected entity in the link module database.

- Delete History: This box lists each value and the corresponding name of the directly selected entity in the link module database that has been deleted from the AutoCAD drawing.

The *Handle in Drawing* edit box can be used to relate the handle in a text box to the graphic entity in the AutoCAD drawing. When the *Show* button is checked, the link module highlights the corresponding entity in the AutoCAD drawing.

The All Entities in Project option has the advantages that users can get an overview of all the changed directly selected values and check whether the correct values have been obtained. The disadvantage is the long time required if the AutoCAD drawing is a complex one, since the link module must compare each handle saved in its database to the handle of every entity in the AutoCAD drawing. In a typical case, it will take about 20 seconds to update 12 entities in the database if the AutoCAD drawing contains 42 entities.

Redefine value

If there is no *handle* property available for the item that needs to be updated, the user must redefine it manually through the user interface shown in Figure 3.15. The redefine value interface also has eight options, similar to that in the *Select mode*.

The different options in the user interface lead to the corresponding functions in the *Select mode* to determine new values for the changed entities.

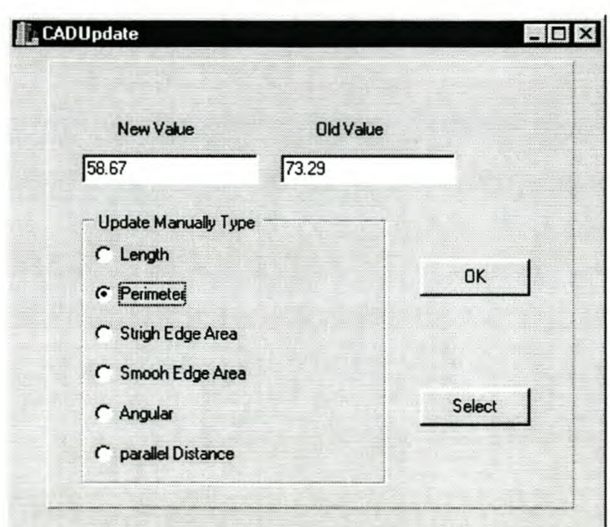


Figure 3.15 Redefine user interface

CHAPTER 4 EVALUATION

4.1 TEST

As a first step to test the link module, it was confirmed that the properties and methods of the following primitive AutoCAD entities could be accessed in an AutoCAD drawing: 3D face, 3D polyline, 3D Solid, Arc, Circle, Line, Polyline, SplyLineLight, Region, Solid and Spline. Each of these has dimensions to test the accuracy of the link module. The entities are shown in Figure 4.1.

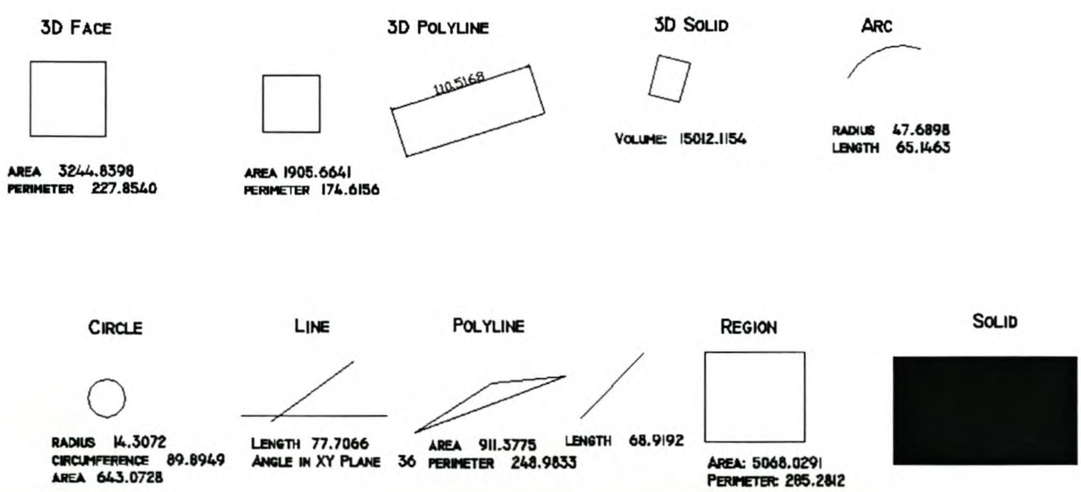


Figure 4.1 Different cases of single entity

The numerical results of the single test of the link module are shown in Table 4.1. compare to the results that inquired in AutoCAD, the direct select results are almost the same value as the AutoCAD dimensions since the link module accesses the AutoCAD entity directly to obtain the results. Table 4.1 also contains the user define method results that will be discussed in the following paragraph.

AutoCAD entity name	Value	Result of direct select method	Result of user Define method	AutoCAD dimension/ inquired result
3D Face	Area	3244.84		3244.8398
	Perimeter		227.85	227.8540
3D Polyline	Area		1905.66	1905.6641
	Length	110.52		110.5168
3D Solid	Volume	15012.12		15012.1154
Arc	Radius	47.69		47.6898
	Area	440.00		
Circle	Radius	14.31		14.3072
	Area	643.07		643.0728
Line	Length	77.71		77.0766
	Angle		35.67	36
Polyline	Area	911.38		911.3775
	Perimeter		248.9833	248.9833
	Length	68.92		68.9192
Region	Area	5068.03		5068.0291
	Perimeter	285.28		285.2812
Solid	Area		442.22	442.2220

Table 4.1 Numerical results of single AutoCAD entity test

Secondly, using the option in the link module, the user-define option was also evaluated. Some tests use the geometric entities in Figure 4.1 and their results are listed in Table 4.1. Parallel distance angle, perimeter, composite line, straight edge area and smooth edge area were tested. The drawing is shown in Figure 4.2, and the numerical results are shown in Table 4.2 (Smooth area value was compared with the value of a circle area).

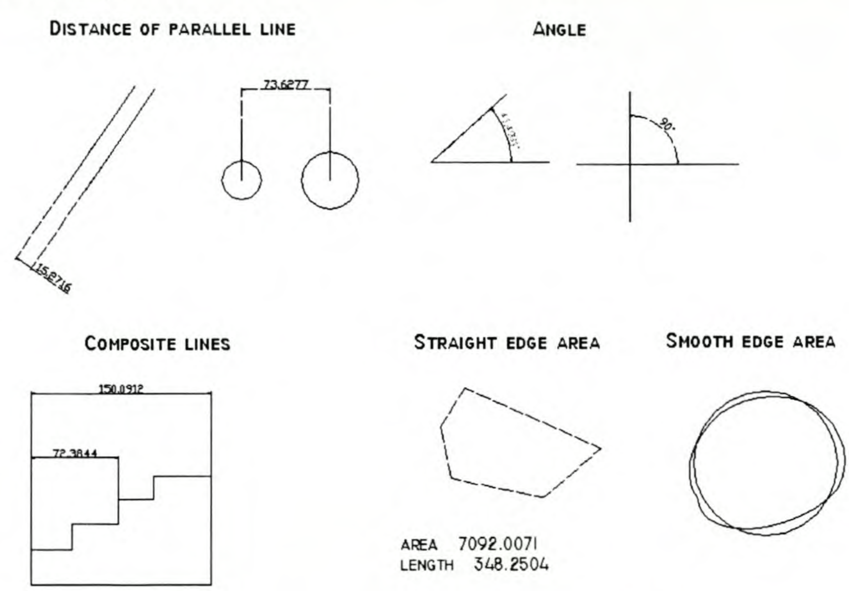


Figure 4.2 User define option examination

AutoCAD entity name	Result of user Define method	AutoCAD dimension/ inquired result
Distance of parallel lines (1)	15.27	15.2716
Distance of parallel lines (2)	73.63	73.6277
Angle (1) (two lines)	41.50	41.4991
Angle (2) (two lines)	90.00	90
Angle (1) (three points)	41.50	41.4991
Angle (2) (three points)	90.00	90
Composite lines	72.38	72.3844
Straight edge area	7092.01	7092.0071
Smooth edge area	10908.68	11259.3567

Table 4.2 Numerical results of the user define option

Compare the results, except the smooth edge area, if a user uses the snap mode to choose the points of the entity, the user can also obtain almost the same values inquired in AutoCAD. For smooth area, accuracy rate is about 96%.

In the perimeter, straight edge area and smooth edge area options, the program limits maximum of ten control points. In addition, the smooth edge area requires that the user define a point as the start point and another point as the end point to define the tangent points of the polyline. All these user selections can contain more than one entity. The program asks the user to choose between the various alternatives through the command line in AutoCAD.

4.2 CASE STUDY

As a case study, a welded assembly was used to evaluate the link module. The assembly is shown in Figure 4.3.

The test was performed mainly using a 2D AutoCAD R14 drawing. The drawing of the part can be seen in Figure 4.4. The dimensions in this drawing can be used to test the accuracy of the selection results in the link module.

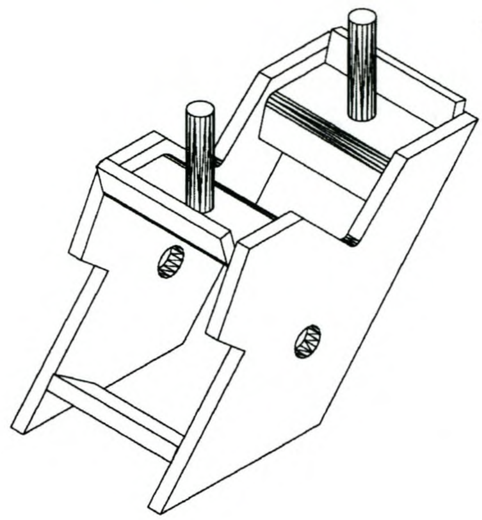


Figure 4.3 A welded product

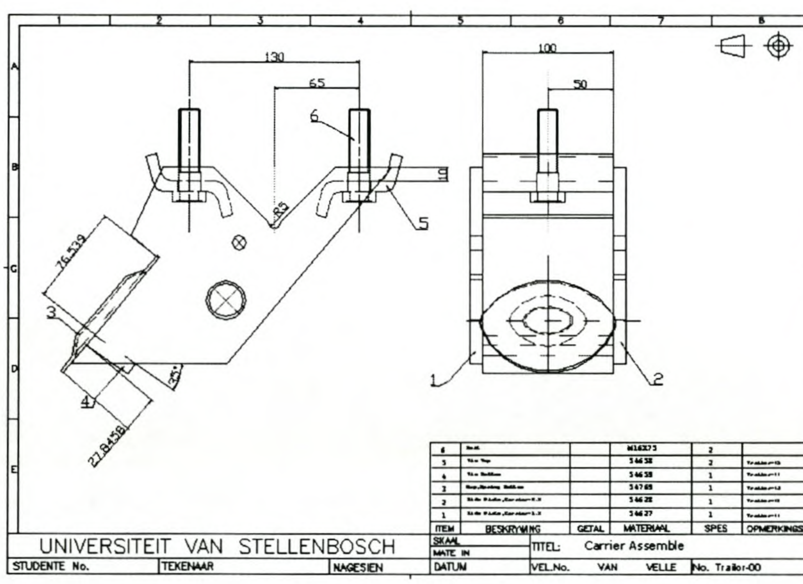


Figure 4.4 Assembly drawing of test example

The case study is performed for each drawing of a part, as shown in Figure 4.5, Figure 4.6 and Figure 4.7. The corresponding numerical results are shown in Table 4.3, Table 4.4 and Table 4.5.

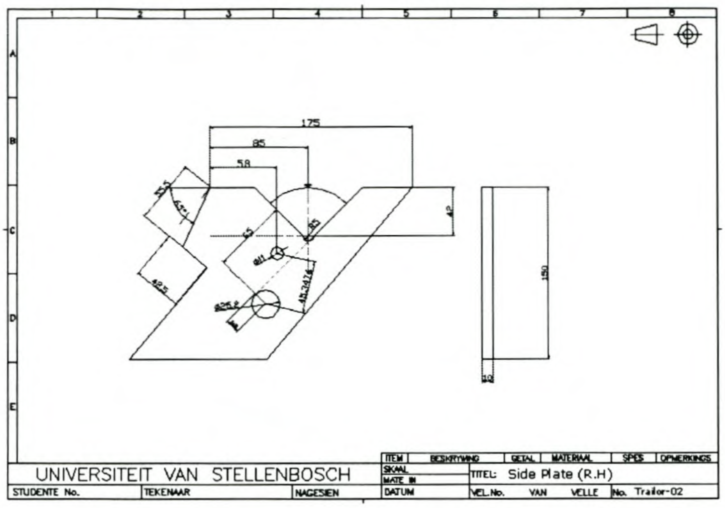


Figure 4.5 Side plate drawing

Process	**Item Name**	**Direct select**	**Manually select**	**AutoCAD**
Flame Cutting Profiles	Right cut Length	195.81	—	195.811
Flame Cutting Profiles	V shape side length	59.4	—	59.4
Flame Cutting Profiles	V Angle	—	90	90
Flame Cutting Profiles	Left cut length segment1	57.46	—	57.463
Flame Cutting Profiles	Left cut angle	—	65.8	65
Flame Cutting Profiles	Left cut angle segment2	27.63	—	27.63
Drilling	Diameter hole1	11	—	11
Drilling	Diameter hole2	25	—	25
Drilling	Arc radius	5	—	5
Drilling	Sheet metal thickness	10	—	10
Drilling	Distance between holes	—	45.34	45.3476

Table 4.3 Numerical results of side plate

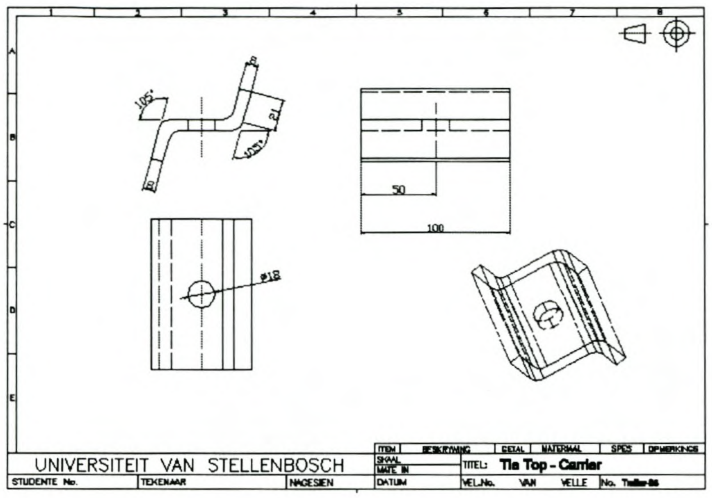


Figure 4.6Tile top drawing

Process	Item Name	Direct select	Manually select	AutoCAD
Bend	Part thickness	8.00		8.00
Bend	Bend angle	—	105.28	105
Bend	Bend perimeter	—	86.72	88.30
Material cost	Part volume	73386.0746	—	73386.07
Drilling	Hole diameter	18	—	18

Table 4.4 Numerical result of tie top

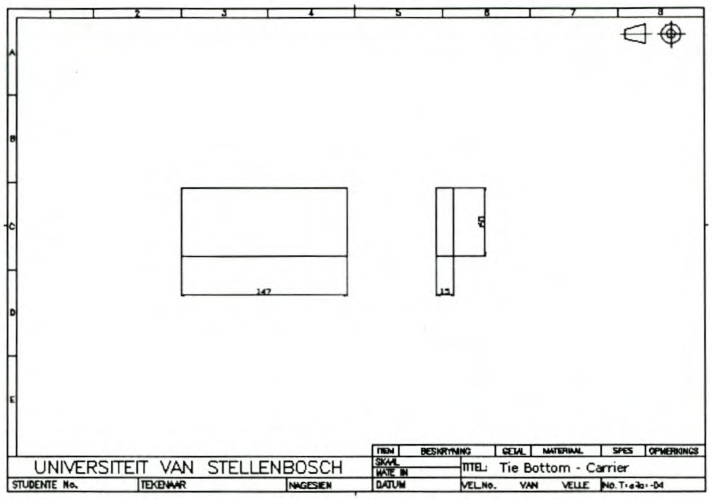


Figure 4.7 Tie bottom drawing

Process	Item Name	Direct select	Manually select	AutoCAD
Sawing	Length	147	—	147
Sawing	width	60	—	60

Table 4.5 The numerical results of tie bottom

The direct select results were exactly correct, since it uses the properties of the AutoCAD geometric entities. The lowest accuracy of the user defined entities was about 96% due to the human inaccuracy of defining points. However, this is still good enough for the conceptual design.

4.3 DEBUG PROBLEM

The main problem in the code debugging is that it often crashes the computer and it is difficult to identify the run-time error. An exception is a application condition that might cause the program to halt or produce incorrect results. C++Builder provides a sophisticated exception handling mechanism and every C++Builder application has default exception handlers that display an error message and try to prevent the program from halting unexpectedly if an exception occurs.

Most of the error messages indicate that the exception has been raised or that the program access is illegal at a certain address and the system remains in a halted condition. An exception is raised potentially due to component methods, run-time library subroutine expressions, hardware faults or assigning an illegal value to a component property. The way in which to handle exception in a program in C++ Builder is to use the *try.. catch..* construct to switch to the catch block code section when an exception arises in the *try..* block. For example, if the program finds that there is no AutoCAD automation available in the thread, it will call the CreateOLEobject function to start the AutoCAD application. If the thread is occupied by an AutoCAD application (such as when AutoCAD is performing a command or waiting for an input), it will show a piece of the message to prompt the user to cancel the operation that AutoCAD is running. But this approach can only work in the execution file of the C++ Builder project and some critical errors can still crash the computer system. Since there are so many OLE function operations, it needs to use too many try.. catch.. functions and this will make the program more difficult to handle.

Problems relating to COM are also the part of the reason for the error generation and error detection difficulties. Since COM is a new and untried technology, the main problems of COM are: [Calvert and Babet, 1999]

- A plug-in or distributed architecture can be prone to bugs because of lack of true compatibility between the different pieces of a program or integrated system.
- A crash happens easily due to careless programming or a failure on Microsoft's part to clearly define the OLE specification. Also, another set of problems will arise if some portion of Microsoft's specification for OLE is proved to be fundamentally flawed, which is not an unthinkable scenario.

This means that critical errors can arise when the application either calls a function without the requisite arguments or misinterprets a return value and no warning is indicated during compilation time. This causes difficulties in defining the parameters of the OLE function. For example, to draw a line in AutoCAD, the Add method for line object needs to use variants as the parameters, instead of simply using three coordinates.

Since only AutoCAD VBA examples are available in AutoCAD, developers need to use VBA code as reference to develop the Borland C++ code. Differences in syntax between VBA and C++ Builder also contribute to the difficulties related to the determination of the OLE function parameters.

Another problem arises from the usage of variant variable. As mentioned before, it requires a considerable amount of computer memory if each inherited object in AutoCAD automation needs to be defined as a separate variant, which increases the amount of the variant variables. This can cause the memory to leak when the user is working with a complex drawing that contains too many graphic objects. The solution here is to free the allocated memory that the variant uses when the related operation is finished.

Running Delphi code in Borland C++ can also give a linker error because of the slight incompatibility between Delphi and Borland C++ Builder. This can be avoided by using "building project" command in Borland C++ Builder whenever changes to the Delphi code were made and using the common variable types that are supported by both C++Builder and Delphi.

CHAPTER 5 CLOSURE

5.1 CONCLUSION

With the competition in the manufacturing industry becoming more and more intense, the demand for economic manufacture as well as competitive products is growing. This strategy increases the need to develop design tools and aids for online cost estimation in the early design stages.

CeDeas is software that was developed by the Department of Mechanical Engineering at the University of Stellenbosch. It is aimed at simple welded structures manufactured in small quantities and focuses on estimating the manufacturing cost of the components and/or assemblies during the late concept design to early detail design stage. The designer can use it to get information such as the material cost, manufacturing cost and total cost of a component or an assembly. The designer can thereby compare the manufacturing costs of alternative designs (e.g. changing the part geometry or using a different manufacturing process) and optimise the design accordingly.

During this phase, the CAD models will typically be incomplete because manufacturing decisions have to be made before all the finer details of a design can be included in the CAD models. Cost estimation during these phases therefore requires inputs from the designer. These inputs are, typically, the intended (or candidate) manufacturing processes and quantitative information about the manufacturing features required for obtaining a cost estimate. Some of the quantitative information will usually be present in the incomplete CAD model.

In order to use CeDeas effectively, product manufacturing information must be communicated between CAD system such as AutoCAD and CeDeas with minimum effort by the users. Many researchers working on the integration of design, process planning, cost information and application programs often use neutral data exchange formats, such as IGES, to transfer information between software applications. These approaches limit the types of information to that provided for in the specifications of the neutral format, It further does not allow for selective updates, such as when a part of a design has been changed and when a corresponding update of the manufacturing cost estimate is required. To overcome these limitations, a link module was developed linking AutoCAD and CeDeas by using COM technology and Borland C++ Builder.

COM is an object-oriented programming model for building software applications made up of modular components. COM allows different software modules, to work together as a single application. COM object creation is language independent. It enables software components to access software services provided by other components, regardless of whether they involve local function calls, operating system calls, or network communications. AutoCAD offers such a flexible COM object with which a user can develop various applications.

By using the link module, a user can easily and accurately get the values of the manufacturing features and send them to CeDeas as parameters to calculate the cost estimation. When the design is adjusted by the designer, it is also easy to update the results by retrieving the database to get the handle property of that entity and recalculate the changed values. The link module has two modes: *Select mode* and *Update Mode*. *Select mode* is used to prompt the user to pick an object in AutoCAD drawing, to save the manufacturing information to the database in CeDeas and to get the manufacturing cost. *Update Mode* is used for getting the cost of a changed design in order to compare the costs of different schemes. The link module can substantially reduce the time needed for the designer to update the geometry-related information and the accuracy is 100% in a direct select model and better than 96% in a user defined model in the case study of a welded production design.

5.2 FUTURE WORK

The link module application is currently developed for AutoCAD R14. In the future, the link module could then be expanded to AutoCAD 2000 or to Mechanical Desktop, both of which have more available automation objects, and it can have more capabilities to deal with 3-D solids. A more detailed understanding of these two sets of AutoCAD automation will be required.

Feature recognition technology can be applied in the link module to assist the process planning. CeDeas aims at simple welded structures manufactured in small quantities.

It can use an object orientation technology to build different modules corresponding to different operations that need welding. It can provide the different manufacturing process costs for the same part to allow the user to select the most effective way for detailed product design.

A significant advantage of the COM technology is that it can provide a means of defining a specification to create a set of non-language specific standard objects. A programmer can develop his/her own automation, which cannot be done using VBA automation in AutoCAD, and then use them for different applications and define the attributes of that automation separately.

CHAPTER 6 REFERENCES

Barton J. J. and Nackman L. R. (1994), Scientific and Engineering C++--An Introduction with Advanced Techniques and Examples, Addison-Wesley, New York

Bernard A. and Brissaud D. (2001), *Manufacturability as a Vector of Design Quality Improvement*, Proceedings International CIRP Design Seminar Design in the New Economy, 6-8, June, Stockholm Sweden, pp. 159-162.

Bidanda B., Kandidal M., and Billo R. E. (1998), *Development of an Intelligent Castability and Cost Estimation System*, International Journal of Production Research, Vol. 36/2, pp. 547-568

Bidarra R. and Bronsvoort W. F. (2000), *Semantic Feature Modelling*, Computer-Aided Design, Vol. 32, pp. 201-225

Bliznakov P. I., Shah J. J. and Urban S. D. (1996), *Integration Infrastructure to Support Concurrence and Collaboration in Engineering Design*, Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, 18-22 Aug, Irvine, California

Boothroyd G., Dewhurst P. and Knight W. (1994), Product Design for Manufacture and Assembly, Marcel Dekker, New York

Borland 2000, C++Builder™, Professional Version 5.0, On-line help, Borland Inprise

Brett B. D., Peters T. J., Demurjian S. A. and Needham D. M. (1996), *Relations Between Features-Prototype Object-Oriented Language Extension on an Industrial Example*, Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, 18-22 Aug, Irvine, California

Brockschmidt K. (1996), Inside OLE, (2nd Edition), in **Microsoft Developer Network Library Studio,** (1998), CD-ROM, Microsoft Press, Washington

Brockschmidt K. (1994), *OLE Integration Technologies: A Technical Overview*, in **Microsoft Developer Network Library Studio,** (1998), CD-ROM, Microsoft Press, Washington

Calvert C. and Babet B. (1999), *The Basic of COM and Type Libraries*, Borland Resdorph K., Gill B. and Roche D., Eds., in C++Builder™ 4 Unleashed, Sams Indianapolis

Chen K., Jiang Z. and Harrison D. K. (2000), *A Generic Integration Approach to the Design and Manufacture of Scroll Compressors*, Proceedings of the 2000 ASME Design Engineering Technical Conference, 10-13 Sep, Baltimore Maryland

Chen Y. M., Miller R. A. and Lu S. C. (1992), *Spatial Reasoning on Form Feature Interactions for Manufacturability Assessment*, Computers in Engineering, Vol. 1, ASME, pp. 29-36

Choi B. K. and Barash M. M. (1985), *STOPP: An Approach to CADCAM Integration*, Computer-Aided Design, May, Vol. 17, No. 4, pp. 162-168

Cottrell S. and Gao J. (2000), *Design for High Pressure Compressor Components Using Object Orientated Technology*, 2000 International CIRP Design Seminar, May 16-18, pp. 289-294

Coyne R. D. and Rosenman M. A. (1990), Knowledge-Based Design Systems, Addison- Wesley, New York

Das A. K. and Langrana N. A. (1996), *Geometry Reconstruction of Vectorized Drawing Consisting of Orthographic Views*, The 1996 ASME Design Engineering Conference and Computers in Engineering Conference, 18-22 Aug, Irvine, California

Davies B. L., Robotham A. J. and Yarwood A. (1986), Computer-Aided Drawing and Design, Chapman & Hall, NewYork

Esawi A. M. K. and Ashby M. F. (1996), *Systematic Process Selection in Mmechanical Design*, Proceedings of the 1996 ASME Design Engineering

Technical Conference and Computers in Engineering Conference, 18-22 Aug, Irvine, California

Fazio T. L. D., Edsall A. C., Gustavson R.E., Hernandez J., Hutchins P. M., Leung H. W., Ludy S.C., Metzinger R. W., Nevins J. L. Tung K. and Whitney D. E. (1993), *A Prototype of Feature-Based Design for Assembly*, Journal of Mechanical Design, December, Vol. 115, pp. 723-733

Feng S. C., Nederbragt W. W. Kaing S. and Sriram R. D. (1999), *Incorporating Process Planning into Conceptual Design*, Proceeding of DETC99 the 1999 ASME Design Engineering Technical Conference, 12-15 Sep, Las Vegas, Nevada

Finger S., Fox M. S., Prinz F. B and Rinderle J. R. (1992), *Concurrent Design*, Applied Artificial Intelligence, Vol. 6, pp. 257-283

Friedman F. L. and Koffman E. B. (1994), <u>Problem Solving, Abstraction, and Design Using C++</u>, Addison-Wesley, New York

Gadh R. and Prinz F. B. (1992), *Manufacture: A framework for Recognizing CAD Interactions in Design-For-Manufacture Analysis*, Advances in Design Automation, Volume. 2, De-Vol. 44-2, ASME, pp. 91-103

Gaines D. M., Castaño F. and Hayes C. (1996), *Reconfigurable Feature Recognition for an Adaptable, Maintainable CAD/CAPP Integration*, Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, 18-22 Aug, Irvine, California

Gaines D. M. and Hayes C. (1999), *A Formative Evaluation of a Feature Extraction and Process Planning Tool*, Proceedings of the 1999 ASME Design Engineering Conference and Computers in Engineering Conference, 12-15 Sep, Las Vegas, Nevada

Ganter M. A. and Skoglund P. A. (1991), *Feature Extraction for Casting Core Development*, Advances in Design Automation, Vol. 1, ASME, De-Vol. 32-1, pp. 93-100

Gao S. and Shah J. J. (1997), *Automatic Recognition of Interacting Machining Features Based on Minimal Condition Sub-Graph*, Proceedings of the 1997 ASME Design Engineering Technical Conference and Computers in Engineering Conference, 14-17 Sep. Sacramento, California

Gibson P., Ismail H. and Sabin M. (1996), *A Feature Recognition Project*, Pratt M. J., Sriram R. D. and Wozny M.J., Eds., in Product Modeling for Computer Integrated Design and Manufacture - International Workshop on Geometric Modeling in Computer Aided Design, 19-23 May, Airlie, Virginia, USA, Chapman&Hall Inc., New York

Groover M. P. and Zimmers E. W. (1984), CAD/CAM: Computer-Aided Design and Manufacturing, Prentice-Hall International, Inc, Englewood Cliffs

Gupta S. K. and Nau D. S. (1995), *Systematic Approach to Analysing the Manufacturability of Machined Parts*, Computer-Aided Design, Vol. 27/5, pp. 323-342

Hill T. M., Bleakman, Earl C., and Miles A. (1994), *Knowledge-Based Integrated Design/Cost System*, Factory 2000-Advanced Automation, Conference Publication, 3-5 October, No. 398, pp. 332-337

Horváth I. H., Kulcsár P. K. and Thernesz V. (1994), *A Uniform Approach to Handling of Feature-Objects in an Advanced CAD System*, Advances in Design Automation Volume 1, ASME, Vol. 69-1/Dec, pp. 547-561

Hundal M. S. and Langholtz L.D. (1992), *Computer-Aided Conceptual Design: An Application of X Windows with C*, DE-Vol. 42, Design Theory and Methodology, ASME, pp. 1-9

Hwang H. J. and Lee S. H. (1999), *A Study Process Representation Using STEP*, Proceedings of the 1999 ASME Design Engineering Conference and Computers in Engineering Conference, 12-15 Sep, Las Vegas, Nevada.

Jayaram S. and Mylebust A. (1990), *Automatic Generation of Geometry Interfaces Between Applications Programs and CADCAM Systems*, Computer-Aided Design, January/February, No.1, Vol. 22, pp. 50-56

Jones P.F. (1992), CAD/CAM: Features, Applications and Management, The Macmillan Press Ltd., London,

Kargas A., Cooley P. and Richards T. H. E. (1988), *Interpretation of Engineering Drawings as Solid Models*, Computer-Aided Engineering Journal, April, pp. 67-78

Kim Y. S. and Wang E. (1999), *Machining Feature Recognition for Cast Then Machined Parts*, Proceedings of the 1999 ASME Design Engineering Conference and Computers in Engineering Conference, 12-15 Sep, Las Vegas, Nevada.

Krause F. L., Gross N., Mezentsev A. A. and Wöhler Th. (2000), *Highly Automated CAD Model Preparation for Downstream Applications*, 2000 International CIRP Design Seminar, 16-18 May

Kumara S. R. T., Kao C. Y., Gallagher M. G. and Kasturi R. (1994), *3-D Interacting Manufacturing Feature Recognition*, Annals of the CIRP, Vol. 43/1, pp. 133-136

Lazaro A. D. S. and Engquist D. T. (1992), *A Knowledge-Based Advisor for the Design of Sheet Metal Parts*, Design for Manufacturability ASME 1992, December, Vol. 51/1992, pp. 35-42

Leibl P., Hundal M. and Hoehne G. (1999), *Cost Calculation with a Feature-Based CAD System Using Modules for Calculation, Comparison and Forecast*, Journal of Engineering Design, Vol. 10/1, pp. 93-102

Li S. G., (1999), AutoCAD Development Technology, China Machine Press

Linthicum D. S. and Klein L. (1994), Using Turbo C++, Que Corporation, Indianapolis

Lippman S. B. (2000), Essential C++, Addison-Wesley Inc., New York

Liu X. D. (2000), *CFACA: Component Framework for Feature-Based Design and Processing Planning*, Computer-Aided Design, Vol. 32/6/2000, pp. 397-408

Mahajan P. V., Poli C., Rosen D. and Wozny M. J. (1993), *Design for Stamping: A Feature-Based Approach*, Design for Manufacturability ASME, De-Vol. 52, pp. 29-49

Math Lecture Group, (1988), Handbook of Mathematics, Science Press, China

Maus R. and Keyes J. (1991), Handbook of Expert Systems in Manufacturing, McGraw-Hill, New York

Maree W. G. (1997), A Cost Estimation Model for Small Production Volumes Masters thesis, University of Stellenbosch

McMahon C. and Browne J. (1998), CAD/CAM: From Principles to Practice, Addison-Wesley Inc., New York

Medland A. J. (1986), The Computer-Based Design Process, Second edition, Chapman & Hall Inc., New York

Nee A. Y. C. and Kumar A. S. (1991), *A Framework for an Object/Rule-Based Automated Fixture Design System*, Annals of the CIRP, Vol. 40/1, pp. 147-151

Newell R. G. and Sancha T. L. (1990), *The Difference Between CAD and GIS*, Vol. 22, Computer-Aided Design, pp. 131-135

Pande S. S. and Palsule N. H. (1988), *GCAPPS - A Computer-Assisted Generative Process Planning System for Turned Components*, Computer-Aided Engineering Journal, Aug, pp. 163-168

Park M. W. and Ko H. (1996), *A Feature Recognition System with Interactive Feature Inspection and Deletion Capabilities*, The International Journal of Advanced Manufacturing Technology, Vol. 12, pp. 190-196

Pedley A. G. and Ehrmann M. (1995), *Explicit Feature Interaction Modeling in CAD and CAM Systems*, proceeding of 11th International Conference on Computer-Aided Production Engineering, 20-21 September 1995, pp. 269-275.

Peng T. K. and Trappey A. J. C. (1995), *User-Friendly Interface Development for CAD-Based Engineering Data Management System*, Anzai Y., Ogagwa K. and

Mori H., Eds., in Symbiosis of Human and Artifact, Elsevier Science, pp. 1153-1159

Petroski H. (1994), Design Paradigms, Cambridge University Press, Cambridge.

Qamhiyah A. Z., Venter R. D. and Benhabib B. (1996), *A Generalized Method for the Classification of Form Features*, Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, 18-22 Aug, Irvine, California

Ransen O. (1997), AutoCAD Programming in C and C++, Wiley & Sons, New York

Regli W. C. and Pratt M. J. (1996), *What Are Feature Interactions?*, Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, 18-22 Aug, Irvine, California

Rohm III T. R., Jones C. L. Tucker S. S. and Jesen C. G. (2000), *Parametric Engineering Design Tools and Applications*, Proceedings of the 2000 ASME Design Engineering Technical Conference and Computers in Engineering Conference, 10-13 Sep, Baltimore Maryland

Schiebler R. and Ehrlenspiel K. (1993), *A Knowledge-Based System as an Integrated Design Assistant*, Dec-Vol. 52, Design for Manufacturability, ASME, pp. 69-74

Schreve K. (1997), Cost Estimating Welded Assemblies Produced in Batches, Masters thesis, University of Stellenbosch

Schuster H. R. (1997), A Computer Aid for Specification Development, Conceptual Design and Manufacturing Cost Estimation in Mechanical Design, Ph.D. thesis, University of Stellenbosch

Shah J. J. and Mantyla M. (1995), Parametric and Feature Based CAD/CAM, John Wiley and sons, New York

Singh N. (1996), Systems Approach to Computer-integrated Design and Manufacturing, Wiley & Sons, New York

Stefano P. D. (1997), *Automatic Extraction of Form Features for Casting,* Computer-Aided Design, Vol. 29/11, pp. 761-770

Stevens G. (1992), A Theoretical and Architectural Specification for an On-Line Real-Time Feature-Based Manufacturability and Cost Comparator (OLRTFBMCC), Masters thesis, University of Rhode Island

Subbaraman M., Paramaguru R., Anand S. and Quo P. C. (1997), *CAD Directed On-Line Cost Estimation Using Activity Based Bosting*, Institute of Industrial Engineers 5th Industrial Engineering Research Conference Proceedings, pp. 781-786

Sun J. H., Ding L. W. and Mi J. (1998), AutoCAD ObjectARX Development Tools and Applications, TsingHua University Press, China

Tönshoff H. K., Aurich J. C. and D'Agostino N. (1996), *A Unified Approach to Free-Form and Regular Feature Modeling*, Annals of the CIRP, Vol. 45/1, pp. 125-128

Tsai J. C. and Chang J. S. (1996), *Development of an AutoCAD-Based Geometric Tolerancing System*, Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, 18-22 Aug, Irvine, California

Ullman D. G. (1997), The Mechanical Design Process, McGraw-Hill, New York

Venuivnod P. K. and Yuen C. F. (1994), *Efficient Automated Geometry Feature Recognition through Feature Code*, Annals of the CIRP, Vol. 43/1, pp. 413-416

Wearring C. (1996), *The Function Feature Model: Bridging the CAD/CAM Gap*, Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, 18-22 Aug, Irvine, California

Wierda L. S. (1991), *Linking Design, Processing Planning and Cost Information by Feature-Based Modelling*, Journal of Engineering Design, Vol. 2/1.

Xiao Y., Long X., Xu J. and Zhang Y. F. (1999), Application Development with C++Builder, China People's Posts and Telecommunications Press, China

Xiao G. and Lee Z. X. (1999), The Principle and Application of Mechanical CAD, TsingHua University Express, China

Zhao Y. and Ridgway K. (1994), *Integrating a Knowledge Based Tool Selection System with Commercial CAD System*, Factory 2000-Advanced Factory Automation, 3-5 Oct, pp. 182-187

Zied I. (1991), CAD/CAM Theory and Practice, McGraw-Hill, New York

Software

AutoCAD (1997), Release 14.01, AutoDesk, Inc., USA

Delphi (1998), Professional, Version 4, Insprise, Corporation, Scouts Valley, CA

Borland C++ Builder (2000), Professional, Version 5, Insprise, Corporation, Scouts Valley, CA

Database Desktop (1996), Version 7.0, Borland International, Inc, Scouts Valley, CA

Borland Database Engine (BDE) Administrator (1998), Version 5.01, Insprise, Corporation, Scouts Valley, CA

APPENDIX A CEDEAS

CeDeas is a module of the design assistant software that was developed by the Department of Mechanical Engineering of the University of Stellenbosch. It aims at simple welded structures manufactured in small quantities and focuses on estimating the manufacturing cost of components and/or assemblies during the late concept design or the early detail design. The designer can use it to get cost information such as the material cost, manufacturing cost and total cost of a component or an assembly. The designer can then compare the manufacturing costs of design alternatives (e.g. changing the part geometry or using different manufacturing process) and optimise the design accordingly.

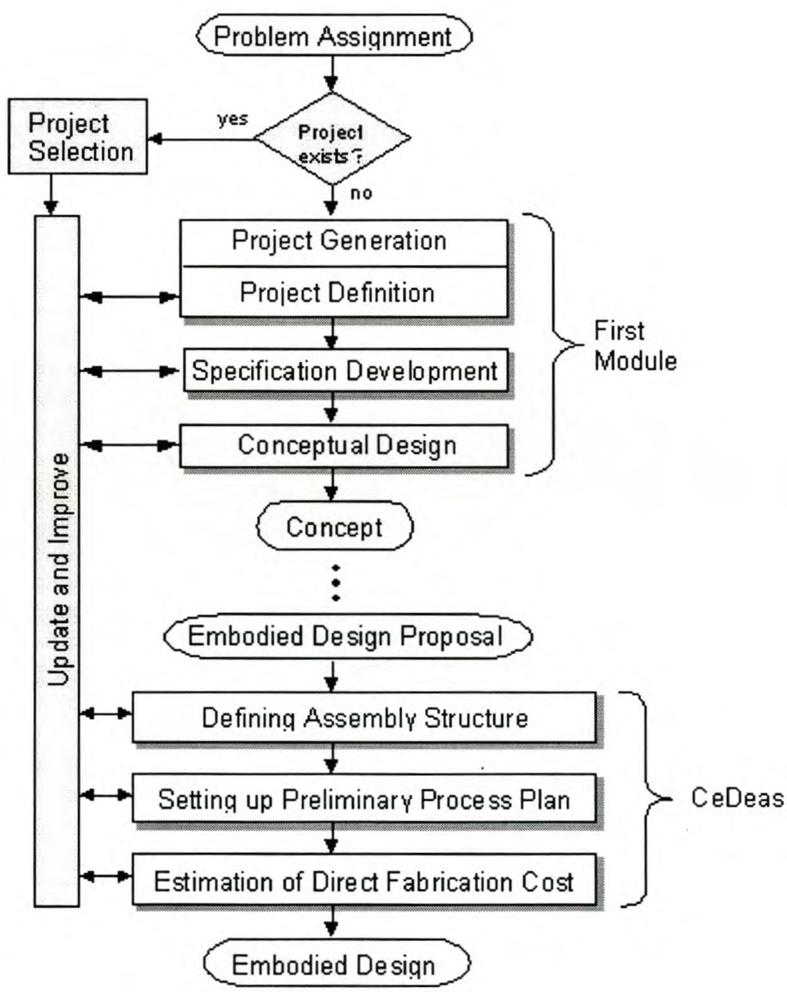


Figure A.1 The Design Assistant flow chart

Figure A.1 shows the flow chart of the design process as implemented in the Design Assistant. The first module supports the designer in the early product development, i.e. the specification and concept development. The second module, CeDeas, provides a framework for manufacturability analysis with an implementation for welded assemblies.

CeDeas uses cost models, which are saved in a database, to estimate the manufacturing cost of components or assemblies. It requires the inputs to get the cost estimation from the designer. These inputs are, typically, the intended (or candidate) manufacturing processes and quantitative information about the manufacturing features required for obtaining a cost estimate. Some of the quantitative information will usually be present in the incomplete CAD model.

The structure of a CeDeas project is illustrated in Figure A.2. The components are regarded as nodes and can be of eight different types: projects, assemblies, parts, library items, outsourced components, processes, setup elements and operation elements.

It uses a tree view to display the hierarchy of nodes in the project. Processes 1 and 2 convert stock material, specified as 'Stock 1' into 'Part 1'. Because every part in this study is manufactured from stock material, the stock material item (the darker block) is not shown in the tree view. The specification of the stock material is handled as part information.

All the procedures can be completed by using the commands in the main menus in the user interface of CeDeas. The times and costs determined for each node are added up to determine the cost of the project.

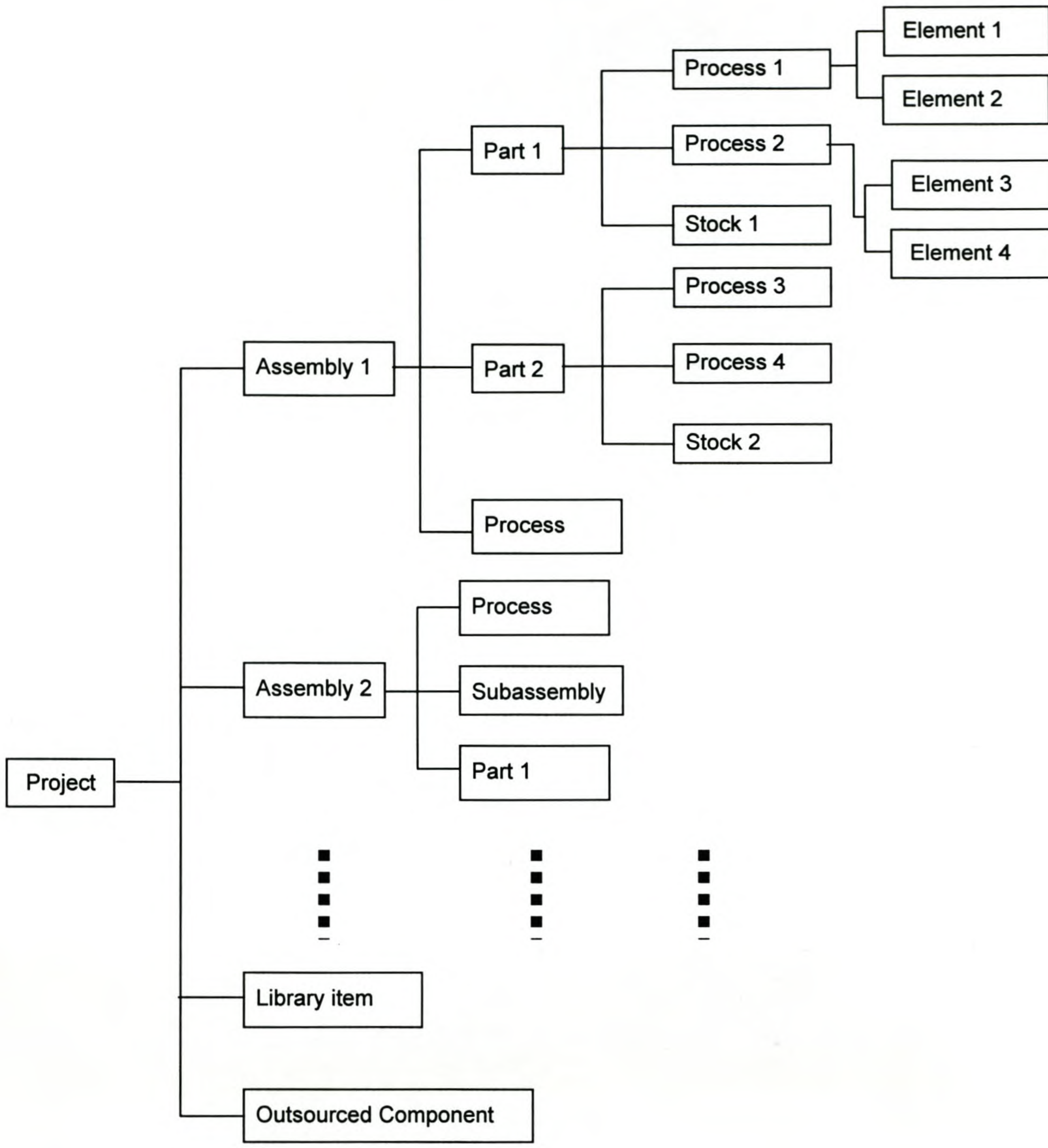


Figure A.2 The project structure

A.1 PROJECT ORGANIZATION

A.1.1 Project Interface

CeDeas starts by either generating a new project or selecting an existing one. When a user selects a project node, the project panel information will appear as shown in Figure A.3. For generating a new project, the user first needs to add a project node into the tree view, enter the name, description and drawing-number, quantity of this assembly. The tree view is used to visually represent the relationship between the nodes of a project. It looks similar to the directory structure in Windows Explorer and each node in the tree view has a glyph that indicates the node type. Child nodes are right indented to show the relationship to the parent node. The structure of the tree will ensure that the quantities of the constituent nodes are added up correctly.

All data subsequently entered will be related to this user-defined project. In the case of more than one project, the projects can be ordered in a hierarchy, i.e., a project can have sub-projects, and so on. This gives the user the opportunity to break down a complex project into smaller sub-projects.

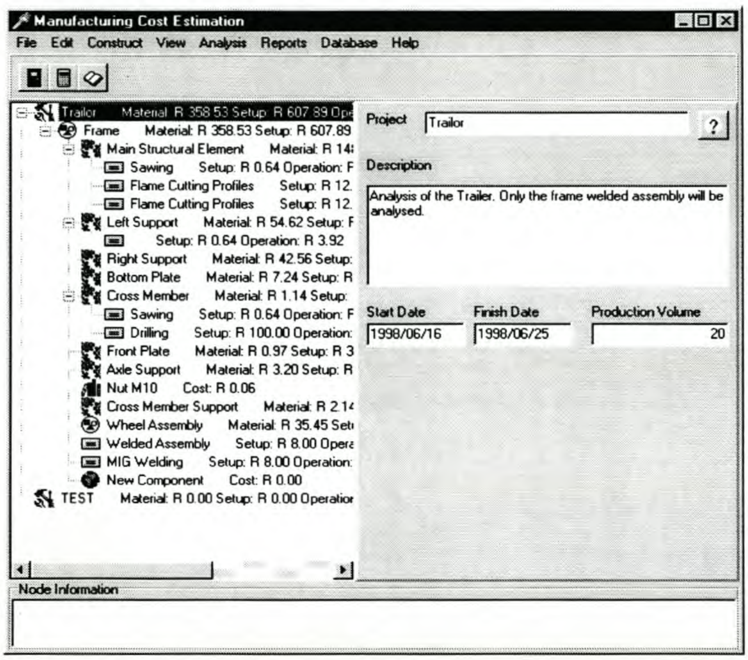


Figure A.3 Work window

A.1.2 Assembly Node

After defining a project, the user can structure the assemblies as nodes of this project. Assembly nodes can have any type of node as a child node except project nodes. They can also be added as children nodes to another assembly node. It then resembles a subassembly.When an assembly node is selected, the assembly information will be displayed in the node information panel that is shown in Figure A.4. It includes assembly name, assembly description, drawing number and Quantity. The drawing number can be referenced to the assembly. The quantity is the number of assemblies that is required to complete one of the parent nodes.

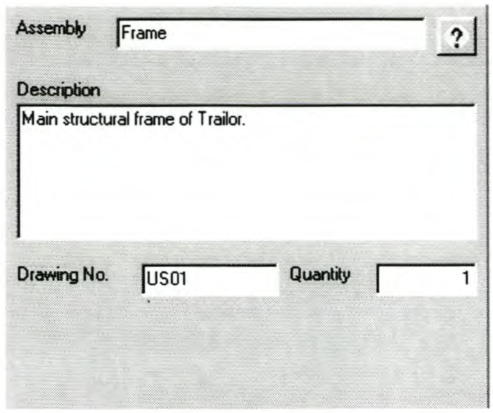


Figure A.4 Assembly node interface

A.1.3 Part Node

When the user structures the parts in an assembly, the node information panel shown in Figure A.5 will appear to let the user give the definition of a part. It includes the part name, part description, drawing number, quantity and part weight and the material description.

When generating a new part, the material needs to be selected first. The material from the database must be assigned to determine the material cost of a part. Nine material classes can be selected. They are engineering alloys, engineering polymers, engineering ceramics, engineering composites, porous ceramics, glasses, woods, elastomers and polymer foams. Engineering alloy class includes alloy steel (free machining), aluminium, copper alloys, low carbon steel, magnesium alloys, medium and high carbon steel, mild steel, nickel alloys, stainless steel, titanium alloys, tool steels and zinc alloys (die cast).

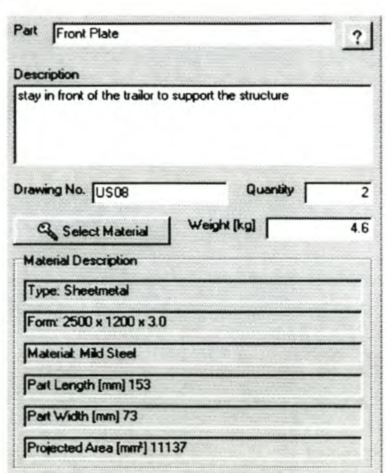


Figure A.5 Part node interface

The Select Material button is used to activate the materials database and to select a material. The material database interface is shown in Figure A.6. The user can enter the material properties and the machining parameters that are used in the cost estimation models for some process operations. A list of the material properties and their units are given in Table A.1. In the material database interface, Vs (the surface generation speed) is the speed at which a machined surface is generated. This is the product of the cutting speed and the feed specified for that material and operation. The unit is square meters per minute. Ps (the specific cutting energy) is the amount of energy required to remove a unit volume of material. The unit is giga-joules per cubic meter.

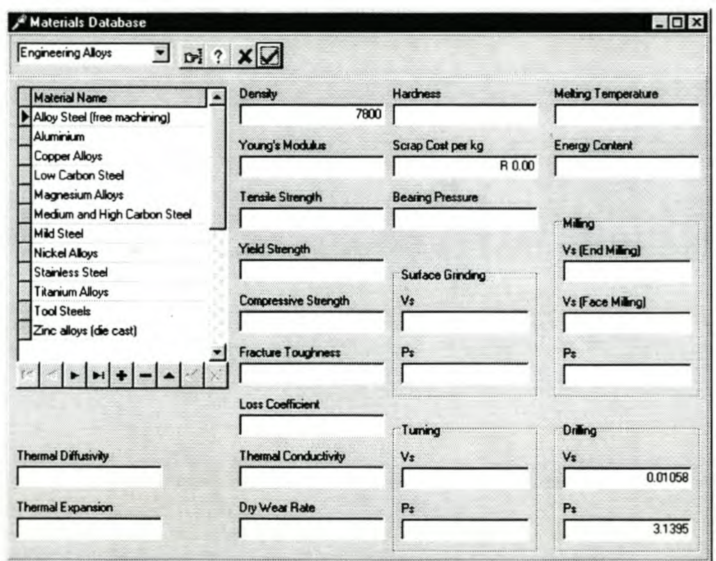


Figure A.6 Material database interface

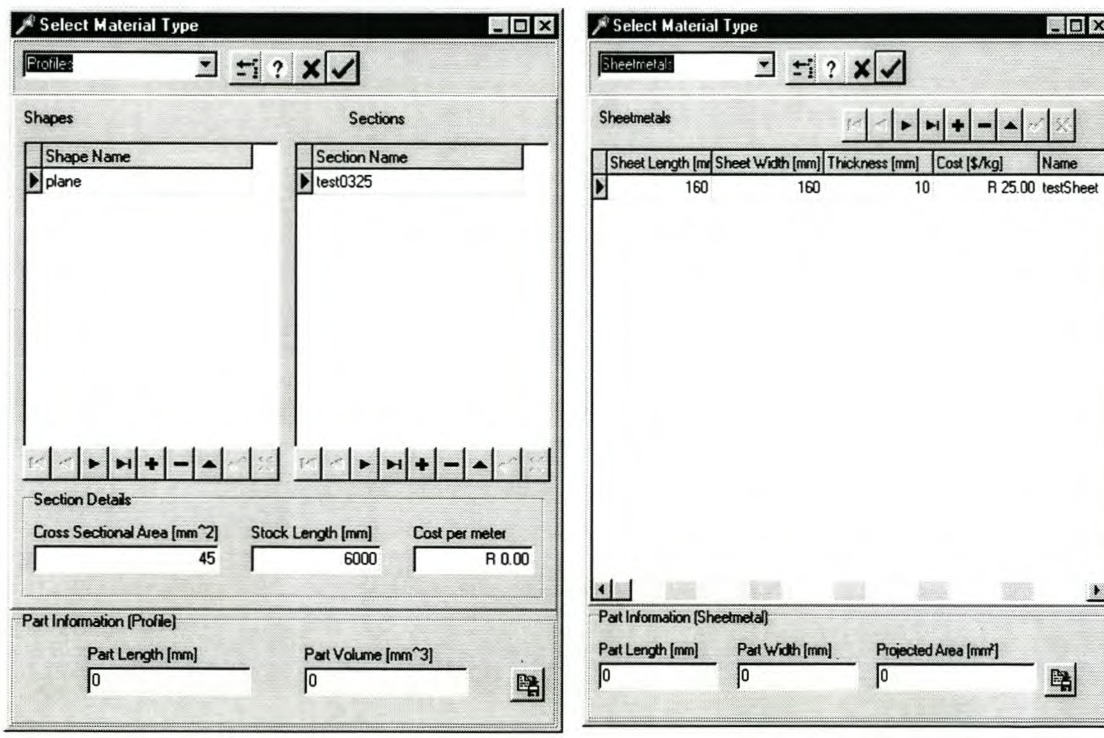
Property name	**Unit**
Scrap Rate	R/kg
Density	Kg/m^3
Young's modulus	GN/ m^2
Tensile Strength	MN/ m^2
Yield Strength	MN/ m^2
Compressive Strength	MN/ m^2
Fracture toughness	MPa/m$^{2/3}$
Loss Coefficient	
Thermal Conductivity	W/mK
Thermal Diffusivity	m^2/s
Thermal Expansion Coefficient	°C^{-1}
Normalized Dry Wear Rate	m^2/N
Maximum Bearing Pressure	MN/ m^2
Energy Content	GJ/ m^2
Hardness	Hb

Table A.1 Material property list

Two types of material forms are supported by the database, profiles and sheet metals. The material types refer to the shape in which the material can be bought. The user-interfaces of these two types of material are shown in Figure A.7.

These two types of material can facilitate the calculation of material costs using parameters associated specifically with a material type, e.g. the sheet metal thickness or the cross sectional area of profiles, in a cost model.

Through the material database interface, user can also add new materials to the material database.



Profile type **Sheet metal type**

Figure A.7 Two material types

A.1.4 Process Node

Process nodes are used to relate manufacturing processes to components or assemblies. Figure A.8 gives the process information panel. It includes process name, description, hour rate, and statistical confidence range and process elements box.

The user can select and add a process into the project. The standard elements will be added automatically as children to the process nodes. They can be viewed by expanding the process nodes. Only process nodes can be added as children to part nodes.

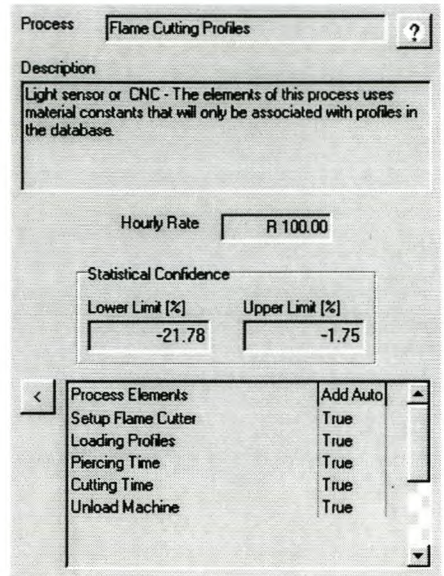


Figure A.8 Process node information panel

The upper and lower limits of the statistical confidence range indicate the variation in the actual manufacturing time that can be expected. It also indicates how well the cost model represents the actual manufacturing time. When a designer must make a selection between two or more concepts, he must take the statistical confidence into consideration. If the time variation of the two concepts overlaps, he must decide whether or not the overlap is significant enough to make a decision based on the manufacturing cost alone.

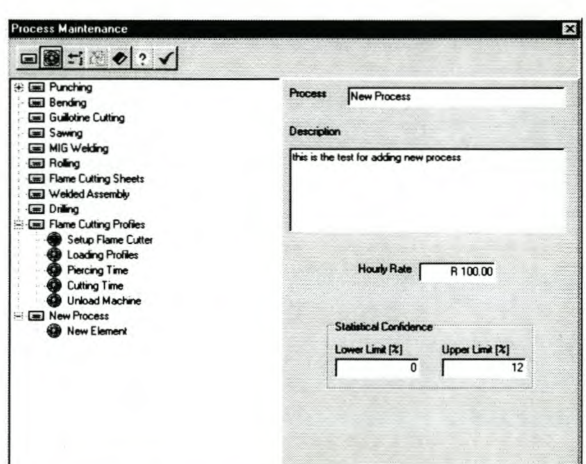


Figure A.9 Process maintenance

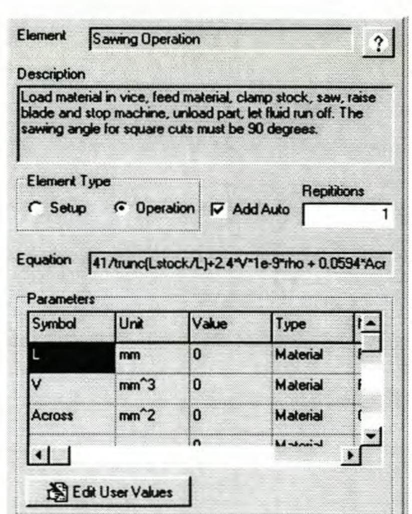
The available processes are:

- Punching
- Bending
- Guillotine cutting
- Sawing
- MIG welding
- Rolling
- Flame cutting sheets
- Welded assembly
- Drilling
- Flame cutting profiles

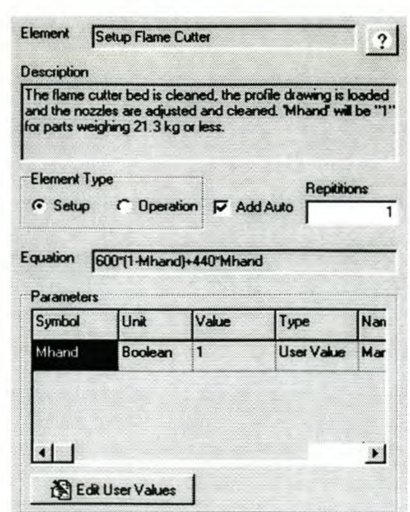
The user can also add new process into the process database via process maintenance interface that is shown in Figure A.9.

A.1.5 Element Node

Element nodes describe the manufacturing steps of a process. Its interface is shown in Figure A.10. There are two types of elements: setup elements and operation elements. Setup elements describe steps that will be repeated only once for all the parts in a batch. Operation elements represent steps of the manufacturing process that will be repeated for every part in a batch. The time associated with setup elements is saved in the setup time field and the time of the operation elements is saved in the operation time field in the database.



Setup interface



Operation interface

Figure A.10 Element interface

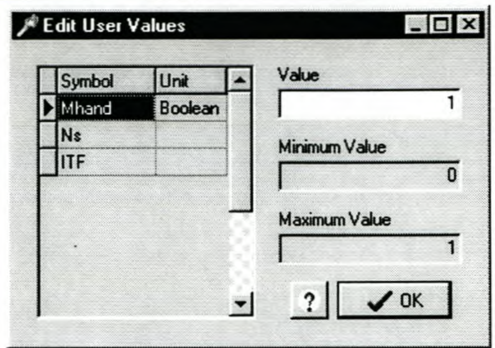


Figure A.11 Edit user values window

The node information panel for both types of elements looks same. It includes the

- Element name.
- Element description.
- Element type.
- The equation that represents the cost model.
- The information about the parameters used in the equation. The parameter symbol, unit, value, type and name are given in this box. Parameter values can be edited by clicking on the edit user values button.
- Repetitions (e.g. the number of times a step must be repeated to complete the manufacturing process).

If the element is a standard element, the add auto box will be checked.

A.1.6 Library Item Nodes

Library item nodes are used to incorporate products in a project that will be bought from a vendor. The library item node information panel is shown in Figure A.12. It includes the catalog name, item, quantity, supplier, supply date, weight, cost and description.

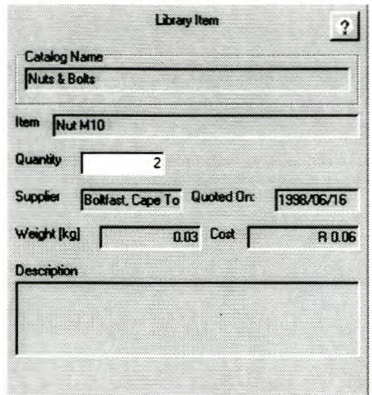


Figure A.12 Library item interface

The user can select a library item from the database or to add a new one and use it in a project. This can be done by editing the library item database through the catalog windows shown in Figure A.13.

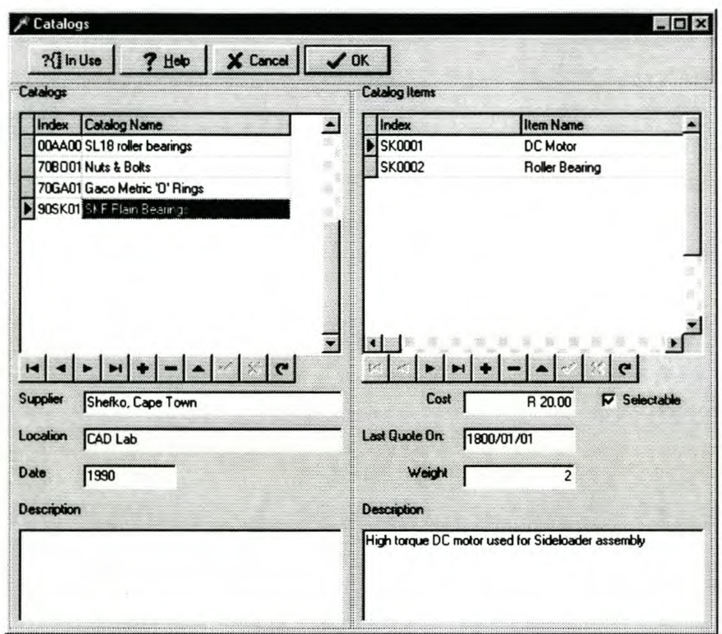


Figure A.13 Catalog window

A.1.7 Outsourced Component Nodes

Outsourced component nodes are used to add products to a project for which a design cost analysis cannot be done and for which a quote has already been received. When an outsourced component node is selected, the node information panel shown in Figure A.14 will appear. It has these items:

- Component name.
- Drawing number.

- Component description.
- Supplier information.
- Information about the quote.
  - A reference number for the quote.
  - The quoted price as a cost per component.
  - Quote date. This is to ensure that the most recent prices are used in the estimation process.

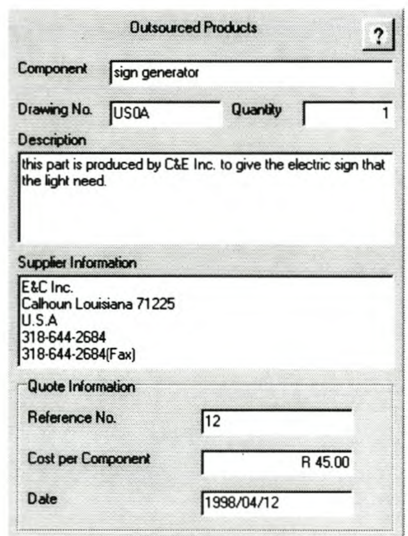


Figure A.14 Outsourced component nodes interface

All these procedures can be finished by using the commands in the main menu of the user interface. CeDeas saves the user input data in each window into the database when the window is closed.

A.2 COST ANALYSIS

A.2.1 Cost/Time Calculation

The user can calculate the times and costs of a selected node after the necessary information have been input. The calculated times and costs represent the time and cost to make one item of the selected node in the case of components or assemblies, or one repetition in the case of element nodes. The total cost and time of all the child nodes are represented in the calculated cost of a parent node.

A.2.2 Cost and Time Contribution Analysis

It is useful to identify the processes and parts that make the most significant contribution to the manufacturing cost and time of a project. A graphical display of the cost and/or time contribution of child nodes can be obtained by doing a cost and time contribution analysis. Only the contribution of the immediate child nodes can be displayed. The costs or times of the nodes are displayed on a bar chart. The nodes are ordered in descending order to facilitate easy recognition of the significant nodes. Design changes that will reduce the cost or time of these expensive nodes will have the most significant impact on the total cost of a project. Seven chart types can be displayed. Read more in the help file on how the costs and times for these charts are calculated.

1. The *Part Cost* chart displays the total manufacturing and material cost.
2. The *Material Cost* chart only displays the material cost.
3. The *Setup Cost* chart only displays the setup cost.
4. The *Operation Cost* chart only displays the operation cost.
5. The *Production Time* chart displays the total production time per part.
6. The *Setup Time* chart displays only the setup time.
7. The *Operation Time* chart displays only the operation time.

A.2.3 Concept Comparison

CeDeas can give a graph to display how the project cost will change with increased production volume. Concepts can be compared if they were analysed as separate projects, i.e. each project is considered a concept.

The cost of tooling, fixtures, patterns, etc. can be added as capital cost in the capital cost field. This cost is recovered over the total production volume. Capital costs should not be estimated in the same project as the product in the tree view. If the tooling is added as a child node to a product's project node, it will be assumed that the number of tooling units will be the same as the specified production volume. Therefore, tooling cost should be estimated in a separate project and added here in the capital cost fields.

A.3 DATABASE

The database architecture of CeDeas is described here. A brief description of the tables will be given. The database class in CeDeas is divided in two separate databases classes: the project database and the materials database.

A.3.1 Project Database

The project database is used to save the structure of a project. Information about the project nodes is saved in separate tables. The different processes available and the parameters needed to calculate the manufacturing cost is saved in this database.

The database architecture is illustrated in Figure A.15. Table A.2 gives the descriptions of these tables.

Table name	Description
Projects	Store the structure of a project.
Tree	Store the tree structure information.
Assemblies	Store the assembly information.
Parts	Store the part information.
LibraryItems	Store items that can be found in catalogues.
Catalogs	Store the information about the catalogue and the supplier of the parts in the catalogue is saved in this table.
Processes	Store the process information.
ElementLinks	Store the links between the tree nodes, the process elements and the parameter values of the elements.
OutSourcedProducts	Store outsourced components information.
ValueLinks	The links to the parameter values are saved in this table. It is necessary to have this table, because two types of parameters can be used in an equation, i.e. material constants and user input parameters.
ProcessElements	Store element information.
Equations	The equation and some information about it are saved in this table.
References	A record of the reference where the equations were found is saved in this table. The reference can be a book, an article, etc.
UserValues	Store the user input values and the link to the parameter.
ParameterList	The list of design parameters is saved in this table along with minimum and maximum allowable value of the parameter.
ParameterNames	The list of the parameter names is saved in this table.
MaterialConstants	This table is the link to the parameter name and its value in the materials database.

Table A.2 Project database tables

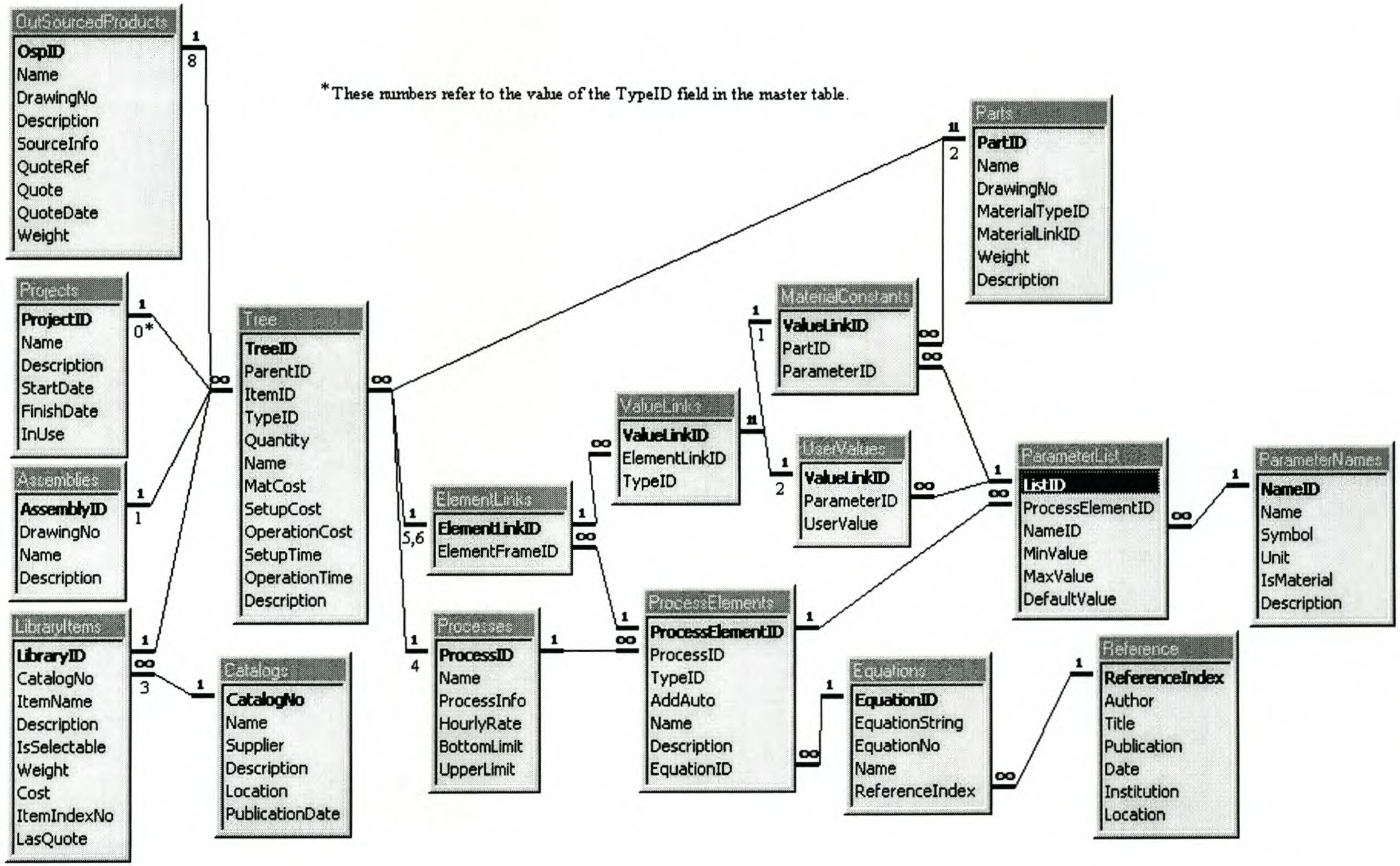


Figure A.15 Project Database Architecture

A.3.2 Materials Database

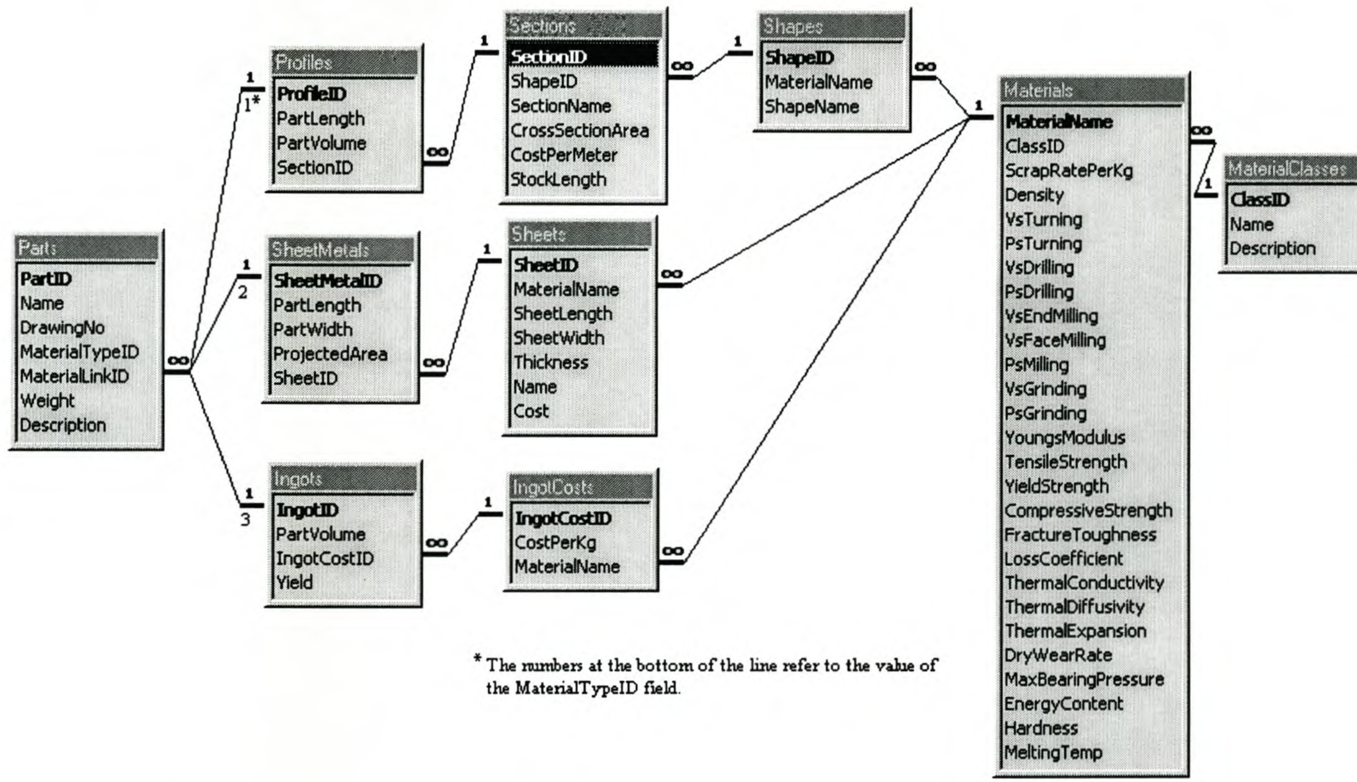
This is not a real separate database. However, because all the information in these tables pertain to the material selected for a part, it is convenient to consider this as a separate database. The Profiles and SheetMetals tables are the link between the part and the material information. The part parameters needed to calculate the material cost are also saved in these tables.

The materials are divided in different types, because the information required for calculations with the materials are related to the type of material used. In this case, "material types" pertain to the shape in which the material is supplied, e.g. standard profiles, sheetmetals, and ingots. This division of material types also makes it possible to incorporate other material types, e.g. ingots and composite materials, at a later stage.

Table A.3 gives the description of these material databases. FigureA.16 shows the Materials Database Architecture.

Field Name	Description
Profiles	The link between parts and the profile type materials is saved in this table. The information about the part required to calculate the profile material cost is also saved in this table.
Sections	Information about the different sections is saved in this table. Angles, channels, round bars; tubes, etc. are all sections.
Shapes	Information about the shape of the profile section is saved in this table.
SheetMetals	The link between a part and a sheet metal is saved in this table. Information needed to calculate the sheet metal cost is also saved in this table.
Sheets	Information about sheet metals that can be found in catalogue s is saved in this table.
MaterialClasses	Store the material class information.
Materials	This table contains all the material parameters. Some of the parameters will be used in estimating the production cost of parts. Others will only be used in selecting materials for a part.

Table A.3 Material databases



FigureA.16 Materials Database Architecture

APPENDIX.B AUTOCAD ENTITY TYPE

EntityType	EntityName
1	ac3DFace
2	ac3DPolyline
3	ac3DSolid
4	acArc
5	acAttribute
6	acAttributeReference
7	acBlockReference
8	acCircle
9	acDimAligned
10	acDimAngular
11	acDimDiametric
12	acDimOrdinate
13	acDimRadial
14	acDimRotated
15	acEllipse
16	acGroup
17	acHatch
18	acLeader
19	acLine
20	acMText
21	acPoint
22	acPolyline
23	acPolylineLight
24	acPolymesh
25	acPviewport
26	acRaster
27	acRay
28	acRegion
29	acShape
30	acSolid
31	acSpline
32	acText
33	acTolerance
34	acTrace
35	acXline

APPENDIX C FUNCTION SOURCE CODE IN THE LINK MODULE

C.1 FUNCTIONS IN SELECT MODE

```
//Head file variable definition
    Variant CADObject,CADDoc,CADDrawing;
    Variant CADSpace,CADPaper;
Variant CADSet, CADUtility;
Variant CADSelect,SelectonScreen,SelectonScreen1;
Variant MySelect,StartPoint,EndPoint;
HWND CADWHandle, AppWHandle;
int EdtNum;   //For different Edit Input
int j1,j2,j3,j4,j8,j9,j10,j11,j12,j13,j14,j15;
int j19,j20,j23,j24,j28,j29,j30,j31,j32,j33,j34;
AnsiString CEName,DrawingNoResult,HandleListL;

//=======================================================
 //Define the Search Function
//=======================================================
void TCADForm::SearchCAD(Variant PCADSelect)
 {
   double SObArea[5][10],SObLength[3][10],SObVolume[10];
   double SObRadius[2][10],SObPerimeter[2][10];
   double SHDistance[12][10];
   double SBoxArea[3][10];
   double TranVaule[10];
   j1=j2=j3=j4=j8=0;
   j19=j23=j24=j28=j30=j31=0;
   TStringList* CADHandleList1 = new TStringList();
   CADHandleList1->Duplicates = dupAccept;
   TStringList* CADHandleList2 = new TStringList();
   CADHandleList2->Duplicates = dupAccept;
   TStringList* CADHandleList3 = new TStringList();
   CADHandleList3->Duplicates = dupAccept;
   TStringList* CADHandleList4 = new TStringList();
   CADHandleList4->Duplicates = dupAccept;
   TStringList* CADHandleList5 = new TStringList();
   CADHandleList5->Duplicates = dupAccept;
   TStringList* CADHandleList6 = new TStringList();
   CADHandleList6->Duplicates = dupAccept;
   TStringList* CADHandleList7 = new TStringList();
   CADHandleList7->Duplicates = dupAccept;
   TStringList* CADHandleList8 = new TStringList();
   CADHandleList8->Duplicates = dupAccept;
```

```
   TStringList* CADHandleList9 = new TStringList();
   CADHandleList9->Duplicates = dupAccept;
   TStringList* CADHandleList10 = new TStringList();
   CADHandleList10->Duplicates = dupAccept;
   TStringList* CADHandleList11 = new TStringList();
   CADHandleList11->Duplicates = dupAccept;
// define the initial values
    for(int j=0;j<10;j++)
   {  for(int m = 0;m<5;m++)
        {SObArea[m][j] = 0; }
      SObLength[0][j] = 0;       SObLength[1][j] = 0;
      SObLength[2][j] = 0;       SObVolume[j]    = 0;
      SObPerimeter[0][j] = 0;       SObPerimeter[1][j] = 0;
      SObRadius[0][j] = 0;       SObRadius[1][j] = 0;
      SBoxArea[0][j] = 0;       SBoxArea[1][j] = 0;
      SBoxArea[2][j] = 0;       TranVaule[j] = 0;
  }
   String CADHandle,CADHandle1,CADHandle2;
   int Bound2[] = {0,11};
   Variant MinPoint,MaxPoint,SolidPoint;
   String CADprompt,UserInput,CADprompt1,
    UserInput1,CADprompt2,UserInput2;
   int typeNum;
   int Count = PCADSelect.OlePropertyGet("Count");
//------------------------------------------------------
  for (int i=0;i<=Count-1;i++)
     { SelectonScreen = PCADSelect.OleFunction("Item",i);
       int selectType = SelectonScreen.OlePropertyGet("EntityType");
       switch (selectType ) {
        //------------ 3D Face  SBoxArea[0][j1] ------------------
        case 1:
          CADHandle = SelectonScreen.OlePropertyGet("Handle");
          CADHandleList1->Add(CADHandle);
          SelectonScreen.OleProcedure("GetBoundingBox",&MinPoint,&MaxPoint);
          CADprompt = "The shape of this projected area is an Square or Retangular area and
parallel X/Y axis? N-No/Y-Yes: ";
          UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
               //let user identify the shape, if suit for the Boundary, then that
               //will be the real area, otherwise the user must choose manually
          if(UserInput == "y"||UserInput == "Y")
            {SBoxArea[0][j1] = GetArea(MinPoint,MaxPoint);}
          else
            {MessageBox(CADWHandle,"Please select Manually","suggestion",MB_OK);
             j1--;}
          j1++;
        break;
//----------------- 3DPolyline SObLength[0][j2] CADHandleList2 ------------
        case 2:
          CADHandle = SelectonScreen.OlePropertyGet("Handle");
```

```
        CADHandleList2->Add(CADHandle);
        CADprompt = "Get the length of this 3DPloyLine? N-No/Y-Yes: ";
        UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
        if( UserInput == "y"||UserInput == "Y")
        { SelectonScreen.OleProcedure("GetBoundingBox",&MinPoint,&MaxPoint);
          CADprompt1 = "This is an straigh line? N-No/Y-Yes: ";
          UserInput = CADUtility.OleFunction("GetString",0,CADprompt1);
            // let user identify the shape, if suit for the Boundary, then that
            // will be the real Length, otherwise the user must choose manually
          if(UserInput == "Y"||UserInput == "y")
           {SObLength[0][j2] = GetLength(MinPoint,MaxPoint); }
           else
           {MessageBox(CADWHandle,"Please select Manually","suggestion",MB_OK);
             //there are two option: goback to original from or call select function here
            j2--; }       }
        j2++;
      break;
//------------------- 3DSoild SObVolume[j3],SBoxArea[1][j3] CADHandleList3-----------
      case 3:
        CADHandle = SelectonScreen.OlePropertyGet("Handle");
        CADHandleList3->Add(CADHandle);
        CADprompt = "Get the Volume or projected area? V-Volume/A-Area: ";
        UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
        if(UserInput == "v"||UserInput == "V")
         { SObVolume[j3] = SelectonScreen.OlePropertyGet("Volume"); }
        else
         { CADprompt1 = "Get the projected area in XY plane-1, XZ-2, YZ-3: ";
           UserInput1 = CADUtility.OleFunction("GetString",0,CADprompt1);
           if (UserInput1 == "1"|| UserInput1 == "2"|| UserInput1 == "3");
           { SelectonScreen.OleProcedure("GetBoundingBox",&MinPoint,&MaxPoint);
             CADprompt2 = "The shape of this projected area is an Square or Retangular area
and parallel X/Y axis? N-No/Y-Yes: ";
             UserInput2 = CADUtility.OleFunction("GetString",0,CADprompt2);
                 // let user identify the shape, if suit for the Boundary, then that
                 // will be the real area, otherwise the user must choose manually
             if(UserInput2 == "y"||UserInput2 == "Y")
              {  SBoxArea[1][j3] = GetArea(MinPoint,MaxPoint, StrToInt(UserInput1));}
             else
              {  MessageBox(CADWHandle,"Please select Manually","suggestion",MB_OK);
                 j3--;   } } }
         j3++;
      break;
//-------------- Arc  SObRadius[0][j4],SObArea[0][j4],CADHandleList4 ------------
      case 4:
        CADHandle = SelectonScreen.OlePropertyGet("Handle");
        CADHandleList4->Add(CADHandle);
        CADprompt = "Get the arc's Radius or Area that this arc contian? R-radius/A-Area: ";
        UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
        // Let user identify what's the feature
```

```
        if(UserInput == "r"||UserInput == "R")
          { SObRadius[0][j4] = SelectonScreen.OlePropertyGet("Radius");}
        else
          { if(UserInput == "a"||UserInput == "A")
              {SObArea[0][j4] = SelectonScreen.OlePropertyGet("Area"); }
            else
              {MessageBox(AppWHandle,"You didn't select anything","warning",MB_OK);
               j4--;  } }
        j4++;
       break;
//--------------- Circle   SObRadius[1][j8],SObAre[1][j8] CADHandleList5 ------------
      case 8:
        CADHandle = SelectonScreen.OlePropertyGet("Handle");
        CADHandleList5->Add(CADHandle);
        CADprompt = "Get the circle's Radius or Area that this arc contian? R-radius/A-Area:
";
        UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
        if(UserInput == "r"||UserInput == "R")
          {  SObRadius[1][j8] = SelectonScreen.OlePropertyGet("Radius"); }
        else
          {   if(UserInput== "a"||UserInput == "A")
               {SObArea[1][j8] = SelectonScreen.OlePropertyGet("Area");  }
              else
               {MessageBox(CADWHandle,"You didn't select anything","warning",MB_OK);
                j8--; } }
        j8++;
      break;
//-------------- Line  SObLength[1][j19] CADHandleList6 ----------------------------
      case 19:
        CADHandle = SelectonScreen.OlePropertyGet("Handle");
        CADHandleList6->Add(CADHandle);
        MinPoint = SelectonScreen.OlePropertyGet("StartPoint");
        MaxPoint = SelectonScreen.OlePropertyGet("EndPoint");
        SObLength[1][j19] = GetLength(MinPoint,MaxPoint);
        j19++;
      break;
//--------- PolyLine SObLength[2][j23],SObPerimeter[0][j23], ------------------------
//------------------ SObArea[2][j23],CADHandleList7 --------------------------------
      case 24:
        CADHandle = SelectonScreen.OlePropertyGet("Handle");
        CADHandleList7->Add(CADHandle);

        CADprompt  =  "Get  the  PolyLine's  Length  or  Area  that  this  Polyline  contians?  L-
Length/A-Area: ";
        UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
             // Let user identify what's the feature
        if(UserInput == "l"||UserInput == "L")     //get the length
         {CADprompt1 = "This is an straigh line? N-No/Y-Yes: ";
          UserInput1 = CADUtility.OleFunction("GetString",0,CADprompt1);
```

```
if( UserInput1 == "y"||UserInput1 == "Y")
 { SelectonScreen.OleProcedure("GetBoundingBox",&MinPoint,&MaxPoint);
   SObLength[2][j23] = GetLength(MinPoint,MaxPoint); }
else     // if it's not a stright line
 { CADprompt2 = "Get the total length? N-No/Y-Yes: ";
   UserInput2 = CADUtility.OleFunction("GetString",0,CADprompt2);
   if(UserInput2 == "Y"||UserInput2 == "y")
    {Variant PLineSegment = SelectonScreen.OleFunction("Coordinates");
     SObPerimeter[0][j23]  = GetTotalLength(PLineSegment); }
   else //give up the total line
      {MessageBox(CADWHandle,"You didn't select anything","warning",MB_OK);
       j23--; } }//End of else
}
else  // get the area
 { if(UserInput == "a"||UserInput == "A")
    {  SObArea[2][j23] = SelectonScreen.OlePropertyGet("Area"); }
   else
    {MessageBox(CADWHandle,"You didn't select anything","warning",MB_OK);
     j23--; }  }
j23++;
break;
//-------------------- SPlyLineLight SObArea[3][j24],CADHandleList8 -----------------
case 23:
  CADHandle = SelectonScreen.OlePropertyGet("Handle");
  CADHandleList8->Add(CADHandle);
  CADprompt  = "you can only get the area that this spline contains,get area? Y-Yes/N-
No: ";
  UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
       // Let user identify what's the feature
  if(UserInput == "y"||UserInput == "Y")     //get the length
   {SObArea[3][j24] = SelectonScreen.OlePropertyGet("Area");}
  else
   {MessageBox(CADWHandle,"You didn't select anything","warning",MB_OK);
    j24--; }
  j24++;
break;
//------------- Region SObPerimeter[1][j28] SObArea[4][j28], CADHandleList9 ---------
case 28:
  CADHandle = SelectonScreen.OlePropertyGet("Handle");
  CADHandleList9->Add(CADHandle);
  CADprompt  = "Get this Region's Perimeter or Area? P-Perimeter/A-Area: ";
  UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
  if(UserInput == "p"||UserInput == "P")
    { SObPerimeter[1][j28] = SelectonScreen.OlePropertyGet("Perimeter"); }
   else
    { if(UserInput == "a"||UserInput == "A")
       {SObArea[4][j28] = SelectonScreen.OlePropertyGet("Area");    }
      else
       {MessageBox(CADWHandle,"You didn't select anything","warning",MB_OK);
```

```
              j28--; } }
          j28++;
        break;
//--------------- Solid SBoxArea[2][j30]  CADHandleList10 ---------------------------
        case 30:
          CADHandle = SelectonScreen.OlePropertyGet("Handle");
          CADprompt = "Only get the area that this planar solid contains? Y-Yes/N-No: ";
          UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
              // Let user identify what's the feature
          if(UserInput == "y"||UserInput == "Y")     //get the length
            { SolidPoint = VarArrayCreate(Bound2,1,varDouble);
              CADHandleList10->Add(CADHandle);
              SolidPoint = SelectonScreen.OlePropertyGet("Coordinates");
              SBoxArea[2][j30] = GetBoxArea(SolidPoint,1);  //Defind the GetSolid Function
              j30++; }
          else
           { MessageBox(CADWHandle,"You didn't select anything","warning",MB_OK); }
          break;
//---------------- SpLine  Property: SObArea[5][j31]  CADHandleList11 ---------------
        case 31:
          UserHandle = SelectonScreen.OlePropertyGet("Handle");
          CADHandleList11->Add(CADHandle);
          CADprompt = "Get the area this SPline contains(only can select 4 points)? N-No/Y-Yes:
";
          UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
          if( UserInput == "y"||UserInput == "Y")
           { SObArea[5][j31] = SelectonScreen.OlePropertyGet("Area");
             j31++; }
          else
            MessageBox(CADWHandle,"You didn't select anything","warning",MB_OK);
        break;
       //--------------------------------------------------------------------------------
        default:
          MessageBox(CADWHandle,"you did not select the standard Geometry Entity and please
select again", "warning",MB_OK);
         } //End of Switch case..
   }          //End of the big For Loop
//-----------------------------------------------------------------------------------
//                   see the different case   AND set uo the other valiues:
//-----------------------------------------------------------------------------------
//---------- 3D-Face j1 Single ------------------------------------------------------
//---------- SBoxArea[0][j1] , CADHandleList1 ---------------------------------------
   if(j1==1)
   {ObArea = SBoxArea[0][0];
    UserHandle = CADHandleList1->Strings[0];
    SetDatabase1(DrawingNoResult,0,1,ObArea,"3DFace","BA",UserHandle,CEName); }
//---------- 3D-Face j1 Multiple ----------------------------------------------------
//---------- SBoxArea[0][j1] , CADHandleList1 ---------------------------------------
   if(j1>1)
```

```
{Variant SelectonScreen1;
       switch(MessageBox(CADWHandle,"Got more than one Projected Area, Add
       all?","Query",MB_YESNO))
 {case 6:       //slect single one
   ObArea = 0;
   for(int n = 0;n<j1;n++)
    {ObArea += SBoxArea[0][n];
     TranVaule[n] = SBoxArea[0][n]; }
   SetDatabase(DrawingNoResult,0,j1,ObArea,"3DFace","BA",CADHandleList1,TranVaule,CEName);
   break;
 case 7:      //select All in this set
   MessageBox(CADWHandle,"Please Identify the Area you have interest","Select",MB_OK);
    for(int i2 = 0;i2<Count;i2++)
     {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
      CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
      for(int h=0;h<j1;h++)
      {CADHandle2 = CADHandleList1->Strings[h];
       if( CADHandle1 == CADHandle2)
       {SelectonScreen1.OleProcedure("Highlight",true);
        CADprompt= "Identify area: N-not and continue/Y-Yes: ";
        UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
        if(UserInput == "y"||UserInput == "Y")
         {ObArea = SBoxArea[0][h];
          UserHandle = CADHandleList1->Strings[h];
          SetDatabase1(DrawingNoResult,0,1,ObArea,"3DFace","BA",UserHandle,CEName);
          i2 = Count+2;
          h = j1+2; }          //Stop the Loop
        if(UserInput == "n"||UserInput == "N")
         {SelectonScreen1.OleProcedure("Highlight",false);} } } }
 delete CADHandleList1;
 break;
 }                  //End of Switch
}                          //End of if(j1>1)
//------------------ 3DPolyline j2 Single ----------------------------------------
//------------------ SObLength[0][j2] CADHandleList2 ------------------------------
if(j2==1)
{ObLength = SObLength[0][0];
 UserHandle = CADHandleList2->Strings[0];
 SetDatabase1(DrawingNoResult,0,1,ObLength,"3DPolyline","L",UserHandle,CEName);
}
//------------------ 3DPolyline j2 Multiple --------------------------------------
//------------------ SObLength[0][j2] CADHandleList2 ------------------------------
if(j2>1){
 Variant SelectonScreen1;
 switch(MessageBox(CADWHandle ,"Got more than one 3DPolyLine Lengths, Add all?",
                   "Query",MB_YESNO))
  {case 6:    //if say yes   //j2
    ObLength = 0;
    for(int n = 0;n<j2;n++)
```

```
        {ObLength += SObLength[0][n];
         TranVaule[n] = SObLength[0][n];}
       UserHandle = "CHandle";

SetDatabase(DrawingNoResult,0,j2,ObLength,"3DPloyline","L",CADHandleList2,TranVaule,CEName);
       break;
     case 7:              //say No
       MessageBox(CADWHandle,"Please choose the Length you have interest","Select",MB_OK);
       for(int i2 = 0;i2<Count;i2++)
        {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
         CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
         for(int h=0;h<j2;h++)
          {CADHandle2 = CADHandleList2->Strings[h];
           if(CADHandle1 ==CADHandle2)
            {SelectonScreen1.OleProcedure("Highlight",true);
             CADprompt= "Identify the Length you're interesting? Y-Yes/N-not and continue: ";
             UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
                  // let user identify the shape, if suit for the Boundary, then that
                  // will be the real area, otherwise the user must choose manually
             if(UserInput == "y"||UserInput == "Y")
              {ObLength = SObLength[0][h];
               UserHandle = CADHandleList2->Strings[h];
               SetDatabase1(DrawingNoResult,0,1,ObLength,"3DPolyline","L",UserHandle,CEName);
               UserHandle = "CHandle";
               h = j2+2;
               i2 = Count+2; }
             if(UserInput == "n"||UserInput == "N")
              {SelectonScreen1.OleProcedure("Highlight",false);} } } }
     break;
     }              //End of switch
   delete CADHandleList2;
   }                         //End of if(j2..)
//-------------------- For 3DSolid  j3 single --------------------------------------
//-------------------- Volume[j3],SBoxArea[1][j3]    CADHandleList3 -----------------
  if(j3==1)
   {if(SObVolume[0]>0)
     {ObVolume = SObVolume[0];
      UserHandle = CADHandleList3->Strings[0];
      SetDatabase1(DrawingNoResult,0,1,ObVolume,"3DSolid","V",UserHandle,CEName);}
    if(SBoxArea[1][0]>0)
     {ObArea = SBoxArea[1][0];
      UserHandle = CADHandleList3->Strings[0];
      SetDatabase1(DrawingNoResult,0,1,ObArea,"3DSolid","BA",UserHandle,CEName);}
  }
//----------------- 3D Solid Multiple ----------------------------------------------
//----------------- 3DSolid j3  Volume[j3],SBoxArea[1][j3]    CADHandleList3 --------
  if(j3>1)
  {  Variant SelectonScreen1;
   //------------------------------  3D Solid j3  Multiple -------------------------
```

```
//------------------------------- Volume[j3] CADHandleList3 ---------------------
  if(SObVolume[1]>0)
    {switch(MessageBox(CADWHandle ,"Got more than one 3DPSolid Volumes, Add
      all?","Query",MB_YESNO))
     {case 6:   //if say yes     3DSolid j3
        ObVolume = 0;
        for(int n = 0;n<j3;n++)
         {ObVolume += SObVolume[n];}
        UserHandle = "CHandle";
        SetDatabase(DrawingNoResult,0,j3,ObVolume,"3DSolid",
                         "V",CADHandleList3,SObVolume,CEName);
        break;
      case 7:     //if say No
        MessageBox(CADWHandle ,"Choose the Volunme you have interest","Select",MB_OK);
        for(int i2 = 0;i2<Count;i2++)
         {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
          CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
          for(int h=0;h<j3;h++)
           {CADHandle2 = CADHandleList3->Strings[h];
            if(CADHandle1 == CADHandle2)
             {SelectonScreen1.OleProcedure("Highlight",true);
              CADprompt  =  "Identify  the  Volume  you're  interesting?  Y-Yes/N-not  and
continue: ";
              UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
              if(UserInput == "y"||UserInput == "Y")
               {ObVolume = SObVolume[h];
                UserHandle = CADHandleList3->Strings[h];
                SetDatabase1(DrawingNoResult,0,1,ObVolume,"3DSolid","V",UserHandle,CEName);
                i2 = Count+2;
                h = j3+2; }
             if(UserInput == "n"||UserInput == "N")
             { SelectonScreen1.OleProcedure("Highlight",false);} } } }
      break;
     }                  //End of  Switch
    }                        //End of  if(SObVolume..)
//--------------------- j3 3DSolid Multiple ------------------------------------------
//--------------------- Projected area SBoxArea[1][j3] CADHandleList3 ---------------
  if(SBoxArea[1][1]>0)
            {switch(MessageBox(CADWHandle ,"Got more than one Projected areas, Add
            all?","Query",MB_YESNO))
     {case 6:   //if say yes
       ObArea = 0;
       for(int n = 0;n<j3;n++)
       { ObArea += SBoxArea[1][n];
         TranVaule[n] = SBoxArea[1][n]; }
       UserHandle = "CHandle";

SetDatabase(DrawingNoResult,0,j3,ObArea,"3DSolid","BA",CADHandleList3,TranVaule,CEName);
      break;
```

```
          case 7:
           MessageBox(CADWHandle ,"Please choose the object with projected area you have interest","Select",MB_OK);
           for(int i2 = 0;i2<Count;i2++)
           { SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
             CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
             for(int h=0;h<j3;h++)
             {  CADHandle2 = CADHandleList3->Strings[h];
                if(CADHandle1 == CADHandle2)
                 {SelectonScreen1.OleProcedure("Highlight",true);
                  CADprompt = "Identify the Porjected Area you're interesting? Y-Yes/N-not and continue: ";
                  UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
                  if(UserInput == "y"||UserInput == "Y")
                   {ObArea = SBoxArea[1][h];
                    UserHandle = CADHandleList3->Strings[h];
                    SetDatabase1(DrawingNoResult,0,1,ObArea,"3DSolid","BA",UserHandle,CEName);
                    i2 = Count+2;
                    h=j3+2;}
                  if(UserInput == "n"||UserInput == "N")
                   {SelectonScreen1.OleProcedure("Highlight",false);} } } }
          break;
         }                    //End of switch
         delete CADHandleList3;
       }                          //End of if(SOb..)
  }                                    //End of if(j3..)
 //------------------- Arc  j4 Single -------------------------------------------------
 //------------------- SObRadius[0][j4] CADHandleList4 ----------------------------
  if(j4 ==1)
  {if(SObRadius[0][0]>0)
    {ObRadius = SObRadius[0][0];
     UserHandle = CADHandleList4->Strings[0];
     SetDatabase1(DrawingNoResult,0,1,ObRadius,"Arc","R",UserHandle,CEName);}
   if(SObArea[0][0]>0)
    {ObArea = SObArea[0][0];
     UserHandle = CADHandleList4->Strings[0];
     SetDatabase1(DrawingNoResult,0,1,ObArea,"Arc","A",UserHandle,CEName); }}
 //------------------- Arc  j4 Multiple ------------------------------------------------
 //------------------- SObArea[0][j4] CADHandleList4 ----------------------------
  if(j4>1)
   {if(SObArea[0][1]>0)
      {switch(MessageBox(CADWHandle ,"You chose more than one Arc Areas, Add all?","Query",MB_YESNO))
        {case 6:    //if say yes
           ObArea = 0;
           for(int n = 0;n<j4;n++)
            { ObArea += SObArea[0][n];
              TranVaule[n] = SObArea[0][n]; }
           UserHandle = "CHandle";
```

```
        SetDatabase(DrawingNoResult,0,j4,ObArea,"Arc","A",CADHandleList4,TranVaule,CEName);
        break;
      case 7:      //if not add, select arc
         MessageBox(CADWHandle ,"Please choose the area you have interest","Select",MB_OK);
         for(int i2 = 0;i2<Count;i2++)
         {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
          CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
          for(int h=0;h<j4;h++)
           {CADHandle2 = CADHandleList4->Strings[h];
            if(CADHandle1 == CADHandle2)
             {SelectonScreen1.OleProcedure("Highlight",true);
              CADprompt= "Identify the Area you're interesting? Y-Yes/N-not and continue: ";
              UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
              if(UserInput == "y"||UserInput == "Y")
               {ObArea = SObArea[0][h];
                UserHandle = CADHandleList4->Strings[h];
                SetDatabase1(DrawingNoResult,0,1,ObArea,"Arc","A",UserHandle,CEName);
                i2 = Count+2;
                h = j4+2;}
              if(UserInput == "n"||UserInput == "N")
                {SelectonScreen1.OleProcedure("Highlight",false);} } } }
       break;
      }                   //End of Switch
    }                       // End of if(SObArea..)
//------------------- Arc  j4 Multiple -----------------------------------------------
//------------------- SObRadius[0][j4]  CADHandleList4 ------------------------------
   if(SObRadius[0][1]>0)
     {switch(MessageBox(CADWHandle ,"Got more than one Arc Radius,chooes the correct
one?","Query",MB_YESNO))
      { case 6:    //if say yes
        MessageBox(CADWHandle ,"Please choose the radius you have interest","Select",MB_OK);
        {CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
         for(int h = 0;h<j4;h++)
          {CADHandle2 = CADHandleList4->Strings[h];
           if(CADHandle1 ==CADHandle2)
            {SelectonScreen1.OleProcedure("Highlight",true);
             CADprompt= "Identify the Radius you're interesting? Y-Yes/N-not and continue:
";
             UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
             if(UserInput == "y"||UserInput == "Y")
               {ObRadius = SObRadius[0][h];
                UserHandle = CADHandleList4->Strings[h];
                SetDatabase1(DrawingNoResult,0,1,ObRadius,"Arc","R",UserHandle,CEName);
                h = j4+2; }
             if(UserInput == "n"||UserInput == "N")
               {SelectonScreen1.OleProcedure("Highlight",false);} } } }
       break;
      }                    // End of Switch
    }                         //End of if(SObRadius..)
```

```
     delete CADHandleList4;
   }                                //End of if(j4..)
//----------------------- Circle j8 Single ------------------------------------------
//-------------------- Properties: SObRadius[1][j8],SObAre[1][j8] CADHandleList5
  if(j8==1)
  {if(SObRadius[1][0]!=0)
    {ObRadius = SObRadius[1][0];
     UserHandle = CADHandleList5->Strings[0];
     SetDatabase1(DrawingNoResult,0,1,ObRadius,"circle","R",UserHandle,CEName);}
   if(SObArea[1][0]!=0)
    {ObArea = SObArea[1][0];
     UserHandle = CADHandleList5->Strings[0];
     SetDatabase1(DrawingNoResult,0,1,ObArea,"circle","A",UserHandle,CEName); }
  }
//----------------------- Circle j8 Multiple ---------------------------------------
//-------------------- Properties: SObRadius[1][j8] CADHandleList5 ------------------
  if(j8>1)
   {if(SObRadius[1][1]>0)
     {switch(MessageBox(CADWHandle ,"Got more than one Radius, Add all?","Query",MB_YESNO))
     {case 6:     //if Yes
        ObRadius = 0;
        for(int n = 0;n<j8;n++)
         {ObRadius += SObRadius[1][n];
          TranVaule[n] = SObRadius[1][n];}
        UserHandle = "CHandle";

SetDatabase(DrawingNoResult,0,j8,ObRadius,"Circle","R",CADHandleList5,TranVaule,CEName);
        break;
      case 7:      //if not add each other
        MessageBox(CADWHandle ,"Select the Radius you have interest","Select",MB_OK);
        for(int i2 = 0;i2<Count;i2++)
        {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
         CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
         for(int h=0;h<j8;h++)
          {CADHandle2 = CADHandleList5->Strings[h];
           if(CADHandle1 == CADHandle2)
            {SelectonScreen1.OleProcedure("Highlight",true);
             CADprompt= "Identify the radius you're interesting? N-not and continue /Y-Yes: ";
             UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
             if(UserInput == "y"||UserInput == "Y")
               {ObRadius = SObRadius[1][h];
                UserHandle = CADHandleList5->Strings[h];
                SetDatabase1(DrawingNoResult,0,1,ObRadius,"Circle","R",UserHandle,CEName);
                i2 = Count+2;
                h = j8+2;  }
             if(UserInput == "n"||UserInput == "N")
               {SelectonScreen1.OleProcedure("Highlight",false); } } }
        }                //End of For i2 loop
        break;
```

```
      }
      delete CADHandleList5;
     }
//---------------------- Circle j8 Multiple ---------------------------------------
//-------------------- Properties: SObArea[1][j8] CADHandleList5 --------------------
    if(SObArea[1][1]>0)
     {switch(MessageBox(CADWHandle ,"Got more than one Area, Add all?","Query",MB_YESNO))
       {case 6:      //if say Yes
          ObArea = 0;
          for(int n = 0;n<j8;n++)
           {ObArea += SObArea[1][n];
            TranVaule[n] = SObArea[1][n]; }
          UserHandle = "CHandle";

SetDatabase(DrawingNoResult,0,j8,ObArea,"Circle","A",CADHandleList5,TranVaule,CEName);
          break;
        case 7:
          MessageBox(CADWHandle ,"Identify the Area you have interest","Select",MB_OK);
          for(int i2 = 0;i2<Count;i2++)
           {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
            CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
            for(int h=0;h<j8;h++)
             {CADHandle2 = CADHandleList5->Strings[h];
              if(CADHandle1 == CADHandle2)
               {SelectonScreen1.OleProcedure("Highlight",true);
                CADprompt= "IdentifSy the Area you're interesting? N-not and continue /Y-Yes:
";

                UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
                if(UserInput == "y"||UserInput == "Y")
                 {ObArea = SObArea[1][h];
                  UserHandle = CADHandleList5->Strings[h];
                  SetDatabase1(DrawingNoResult,0,1,ObArea,"Circle","A",UserHandle,CEName);
                  i2 = Count+2;
                  h=j8+2;}
                if(UserInput == "n"||UserInput == "N")
                 {SelectonScreen1.OleProcedure("Highlight",false);} } } }
        break;
      }                              //End of Switch
    }                                   //End of if(SObArea..)
   delete CADHandleList5;
  }                                           //End of if(j8..)
//------------ Line  j19  Single ---------------------------------------------
//------------ SObLength[1][j19]    CADHandleList6 ----------------------------
  if(j19==1)
   {ObLength = SObLength[1][0];
    UserHandle = CADHandleList6->Strings[0];
    SetDatabase1(DrawingNoResult,0,1,ObLength,"Line","L",UserHandle,CEName);}
//------------ Line  j19  Multiple ---------------------------------------------
//------------ SObLength[1][j19]    CADHandleList6 ----------------------------
```

```
  if(j19>1)
   {if(SObLength[1][1]>0)
    {switch(MessageBox(CADWHandle,"Got more than one Lengths, Add all?","Query",MB_YESNO))
      {case 6:      //if say Yes
         ObLength = 0;
         for(int n = 0;n<j19;n++)
          {ObLength += SObLength[1][n];
           TranVaule[n] = SObLength[1][n]; }
         UserHandle = "CHandle";
        SetDatabase(DrawingNoResult,0,j19,ObLength,"Line","L",CADHandleList6,TranVaule,CEName);
         break;
       case 7:         // For Line  j19     Proprety: SObLength[1][j19]   CADHandleList6
         MessageBox(CADWHandle ,"Select the Length you have interest","Select",MB_OK);
         for(int i2 = 0;i2<Count;i2++)
          {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
           CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
           for(int h=0;h<j19;h++)
            {CADHandle2 = CADHandleList6->Strings[h];
             if(CADHandle1 == CADHandle2)
              {SelectonScreen1.OleProcedure("Highlight",true);
               CADprompt= "Identify the Length you're interesting? N-not and continue /Y-Yes:
";

               UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
               if(UserInput == "y"||UserInput == "Y")
                {ObLength = SObLength[1][h];
                 UserHandle = CADHandleList6->Strings[h];
                 SetDatabase1(DrawingNoResult,0,1,ObLength,"Line","L",UserHandle,CEName);
                 i2 = Count+2;
                 h=j19+2; }
                if(UserInput == "n"||UserInput == "N")
                 {SelectonScreen1.OleProcedure("Highlight",false);                      }}}}
//End of For i2 loop
         break;
      }                              //End of switch
    }                                      //End of if(SObLength..)
    delete CADHandleList6;
   }                                          //End of if(j19>1)
//---------------- PloyLine j23 Single -------------------------------------------
//--------- SObPerimeter[0][j23],SObLength[2][j23],SObArea[2][j23] CADHandleList7 --
   if(j23==1)
    {if(SObPerimeter[0][0]>0)
      {ObPerimeter = SObPerimeter[0][0];
       UserHandle = CADHandleList7->Strings[0];
       SetDatabase1(DrawingNoResult,0,1,ObPerimeter,"PolyLine","PL",UserHandle,CEName);}
     if(SObLength[2][0]>0)
      {ObLength = SObLength[2][0];
       UserHandle = CADHandleList7->Strings[0];
       SetDatabase1(DrawingNoResult,0,1,ObLength,"PolyLine","L",UserHandle,CEName);}
     if(SObArea[2][0]>0)
```

```
      {ObArea = SObArea[2][0];
       UserHandle = CADHandleList7->Strings[0];
       SetDatabase1(DrawingNoResult,0,1,ObArea,"PolyLine","A",UserHandle,CEName);} }
//---------------- PloyLine j23 Multiple -----------------------------------------
//---------- SObPerimeter[0][j23] CADHandleList7 ----------------------------------
   if(j23>1)
    {if(SObPerimeter[0][1]>0)
     {switch(MessageBox(CADWHandle,"Got more than one Length, Add all?","Query",MB_YESNO))
       {case 6:      //if say Yes
          ObPerimeter = 0;
          for(int n = 0;n<j23;n++)
           {ObPerimeter += SObPerimeter[0][n];
            TranVaule[n] = SObPerimeter[0][n]; }
          UserHandle = "CHandle";

SetDatabase(DrawingNoResult,0,j23,ObPerimeter,"PolyLine","PL",CADHandleList7,TranVaule,CEName);
          break;
        case 7:          //j23 Properties: SObLength[2][j23],SObArea[2][j23]  CADHandleList7
          MessageBox(CADWHandle ,"Identify the PolyLength you have interest","Select",MB_OK);
          for(int i2 = 0;i2<Count;i2++)
           {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
            CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
            for(int h=0;h<j23;h++)
              {CADHandle2 = CADHandleList7->Strings[h];
               if(CADHandle1 == CADHandle2)
                {SelectonScreen1.OleProcedure("Highlight",true);
                 CADprompt= "Identify Length you're interesting? N-not and continue /Y-Yes: ";
                 UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
                 if(UserInput == "y"||UserInput == "Y")
                   {ObPerimeter = SObPerimeter[0][h];
                    UserHandle = CADHandleList7->Strings[h];

SetDatabase1(DrawingNoResult,0,1,ObPerimeter,"PolyLine","PL",UserHandle,CEName);
                    i2 = Count+2;
                    h=j23+2; }
                 if(UserInput == "n"||UserInput == "N")
                   {SelectonScreen1.OleProcedure("Highlight",false);} } } }
        break;
         }                              //End of switch
       }                                      //End of if if(SObPerimeter..)
//---------------- PloyLine j23 Multiple ------------------------------------------
//---------------- SObLength[2][j23] CADHandleList7 ------------------------------
    if(SObLength[2][1]>0)
     {switch(MessageBox(CADWHandle,"Got more than one Perimeter, Add all?","Query",MB_YESNO))
       {case 6:      //if say Yes
          ObLength = 0;
          for(int n = 0;n<j23;n++)
           {ObLength += SObLength[2][n];
            TranVaule[n] = SObLength[2][n];}
```

```
          UserHandle = "CHandle";

SetDatabase(DrawingNoResult,0,j23,ObLength,"PolyLine","PL",CADHandleList7,TranVaule,CEName);
          break;
       case 7:
         MessageBox(CADWHandle ,"Identify the Perimeter you have interest","Select",MB_OK);
          for(int i2 = 0;i2<Count;i2++)
           { SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
             CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
             for(int h=0;h<j23;h++)
              { CADHandle2 = CADHandleList7->Strings[h];
                if(CADHandle1 == CADHandle2)
                 {SelectonScreen1.OleProcedure("Highlight",true);
                  CADprompt= "Identify Length you're interesting? N-not and continue /Y-Yes:
";

                  UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
                  if(UserInput == "y"||UserInput == "Y")
                    {ObLength = SObLength[0][h];
                     UserHandle = CADHandleList7->Strings[h];

SetDatabase1(DrawingNoResult,0,1,ObLength,"PolyLine","PL",UserHandle,CEName);
                     i2 = Count+2;
                     h=j23+2; }
                  if(UserInput == "n"||UserInput == "N")
                    {SelectonScreen1.OleProcedure("Highlight",false); } } } }
         break;
        }                            //End of switch
      }                                    //End of if if(SObLength..)
//---------------- PloyLine j23 Single --------------------------------------------
//---------------- SObArea[2][j23] CADHandleList7 -----------------------------------
   if(SObArea[2][1]>0)
    {switch(MessageBox(CADWHandle ,"Got more than one Area, Add all?","Query",MB_YESNO))
      {case 6:      //if say Yes
         ObArea = 0;
         for(int n = 0;n<j23;n++)
          {ObArea += SObArea[2][n];
           TranVaule[n] = SObArea[2][n]; }
         UserHandle = "CHandle";

SetDatabase(DrawingNoResult,0,j23,ObArea,"PolyLine","A",CADHandleList7,TranVaule,CEName);
         break;
       case 7:
         MessageBox(CADWHandle ,"Identify the Area you have interest","Select",MB_OK);
         for(int i2 = 0;i2<Count;i2++)
          {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
           CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
           for(int h=0;h<j23;h++)
            {CADHandle2 = CADHandleList7->Strings[h];
             if(CADHandle1 == CADHandle2)
```

```
            {SelectonScreen1.OleProcedure("Highlight",true);
             CADprompt= "Identify the Area you're interesting? N-not and continue /Y-Yes: ";
             UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
             if(UserInput == "y"||UserInput == "Y")
              {ObArea = SObArea[2][h];
               UserHandle = CADHandleList7->Strings[h];
               SetDatabase1(DrawingNoResult,0,1,ObArea,"PolyLine","A",UserHandle,CEName);
               i2 = Count+2;
               h=j23+2; }
             if(UserInput == "n"||UserInput == "N")
              {SelectonScreen1.OleProcedure("Highlight",false);} } } }
      break;
    }                 //End of switch
   }                    //End of if(SObArea..)
  delete CADHandleList7;
  }                     // End of  if(j23..)
//---------------------- SPloyLine j24 Single ------------------------------------
//---------------- SObArea[3][j24]   CADHandleList8 --------------------------------
   if(j24==1)
    {ObArea = SObArea[3][0];
     UserHandle = CADHandleList8->Strings[0];
     SetDatabase1(DrawingNoResult,0,1,ObArea,"SPolyLine","A",UserHandle,CEName);}
//---------------------- SPloyLine j24 Multiple ----------------------------------
//---------------- SObArea[3][j24]   CADHandleList8 --------------------------------
   if(j24>1)
   {if(SObArea[3][1]>0)
    {switch(MessageBox(CADWHandle ,"Got more than one Area, Add all?","Query",MB_YESNO))
      {case 6:     //if say Yes
        ObArea = 0;
        for(int n = 0;n<j24;n++)
         {ObArea += SObArea[3][n];
          TranVaule[n] = SObArea[3][n];}

SetDatabase(DrawingNoResult,0,j24,ObArea,"SPolyLine","A",CADHandleList8,TranVaule,CEName);
        break;
      case 7:         //if No     SPloyLine j24  Property: SObArea[3][j24]   CADHandleList8
        MessageBox(CADWHandle ,"Identify the Area you have interest","Select",MB_OK);
        for(int i2 = 0;i2<Count;i2++)
         {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
          CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
          for(int h=0;h<j24;h++)
           {CADHandle2 = CADHandleList8->Strings[h];
            if(CADHandle1 == CADHandle2)
            {SelectonScreen1.OleProcedure("Highlight",true);
             CADprompt= "Identify the area you're interesting? N-not and continue /Y-Yes: ";
             UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
             if(UserInput == "y"||UserInput == "Y")
              {ObArea = SObArea[3][h];
               UserHandle = CADHandleList8->Strings[h];
```

```
            SetDatabase1(DrawingNoResult,0,1,ObArea,"SPloyLine","A",UserHandle,CEName);
            i2 = Count+2;
            h=j24+2; }
          if(UserInput == "n"||UserInput == "N")
           {SelectonScreen1.OleProcedure("Highlight",false);} } } }
      break;
   }                          //End of switch
  }                              //End of if(SObArea..)
 delete CADHandleList8;
 }                                    //End of  if(j24..)
//--------------- Region j28 Single ---------------------------------------------
//------------- SObPerimeter[1][j28] SObArea[4][j28], CADHandleList9 ----------------
 if(j28==1)
  {if(SObPerimeter[1][0]>0)
    {ObPerimeter = SObPerimeter[1][0];
     UserHandle = CADHandleList9->Strings[0];
     SetDatabase1(DrawingNoResult,0,1,ObPerimeter,"Region","PL",UserHandle,CEName); }
   if(SObArea[4][0]>0)
    {ObArea = SObArea[4][0];
     UserHandle = CADHandleList9->Strings[0];
     SetDatabase1(DrawingNoResult,0,1,ObArea,"Region","A",UserHandle,CEName); } }
//--------------- Region j28 Multiple -------------------------------------------
//------------- SObArea[4][j28], CADHandleList9 -----------------------------------
 if(j28>1)
  {if(SObArea[4][1]>0)
   {switch(MessageBox(CADWHandle    ,"Got    more    than    one    Area,    Add    them    each
other?","Query",MB_YESNO))
    {case 6:     //if say Yes
      ObArea = 0;
      for(int n = 0;n<j28;n++)
       {ObArea += SObArea[4][n];
        TranVaule[n] = SObArea[4][n];}
      UserHandle = "CHandle";
      SetDatabase(DrawingNoResult,0,j28,ObArea,"Region","A",CADHandleList9,TranVaule,CEName);
      break;
     case 7:       //if No
      MessageBox(CADWHandle ,"Identify the Area you have interest","Select",MB_OK);
      for(int i2 = 0;i2<Count;i2++)
       {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
        CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
        for(int h=0;h<j28+1;h)
         {CADHandle2 = CADHandleList9->Strings[h];
          if(CADHandle1 == CADHandle2)
           {SelectonScreen1.OleProcedure("Highlight",true);
            CADprompt= "Is this Area you're interesting? N-not and continue /Y-Yes: ";
            UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
            if(UserInput == "y"||UserInput == "Y")
             {ObArea = SObArea[4][h];
              UserHandle = CADHandleList9->Strings[h];
```

```
                SetDatabase1(DrawingNoResult,0,1,ObArea,"Region","A",UserHandle,CEName);
                i2 = Count+2;
                h=j28+2; } } } }
      break;
    }                          //End of switch
   }                              //End of if(SObArea..)
//--------------- Region j28 Multiple ------------------------------------------------
//------------- SObPerimeter[1][j28], CADHandleList9 --------------------------------
    if(SObPerimeter[1][1]>0)
     {switch(MessageBox(CADWHandle ,"Got more than one Perimeter, Add them each
       other?","Query",MB_YESNO))
      {case 6:      //if say Yes
        ObPerimeter = 0;
        for(int n = 0;n<j28;n++)
         {ObPerimeter += SObPerimeter[1][n];
          TranVaule[n] = SObPerimeter[1][n]; }
        UserHandle = "CHandle";

SetDatabase(DrawingNoResult,0,j28,ObPerimeter,"Region","PL",CADHandleList9,TranVaule,CEName);
        break;
      case 7:        //if No
       MessageBox(CADWHandle ,"Identify the Area you have interest","Select",MB_OK);
       for(int i2 = 0;i2<Count;i2++)
        {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
         CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
         for(int h=0;h<j28;h++)
          {CADHandle2 = CADHandleList9->Strings[h];
           if(CADHandle1 == CADHandle2)
            {SelectonScreen1.OleProcedure("Highlight",true);
             CADprompt= "Is this Area you're interesting? N-not and continue /Y-Yes: ";
             UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
             if(UserInput == "y"||UserInput == "Y")
               {ObPerimeter = SObPerimeter[1][h];
                UserHandle = CADHandleList9->Strings[h];
                SetDatabase1(DrawingNoResult,0,1,ObPerimeter,"Region","PL",UserHandle,CEName);
                i2 = Count+2;
                h=j28+2; } } } }
       break;
    }                             //End of switch
   }                                 //End of if(SObPerimeter..)
   delete CADHandleList9;
  }               //End of if(j28..)
//---------------- Solid j30 Single ---------------------------------------------
//-------------- SBoxArea[2][j30]  CADHandleList10 ---------------------------------
  if(j30==1)
   {ObArea = SBoxArea[2][0];
    UserHandle = CADHandleList10->Strings[0];
    SetDatabase1(DrawingNoResult,0,1,ObArea,"Solid","BA",UserHandle,CEName); }
//---------------- Solid j30 Multiple ---------------------------------------------
```

```
//--------------- SBoxArea[2][j30]  CADHandleList10 ---------------------------------
  if(j30>1)
   {if(SBoxArea[2][1]>0)
    {switch(MessageBox(CADWHandle ,"Got more than one Projected Area, Add them each
     other?","Query",MB_YESNO))
     {case 6:     //if say Yes
        ObArea = 0;
        for(int n = 0;n<j30;n++)
         {ObArea+= SBoxArea[2][n];
          TranVaule[n] = SBoxArea[2][n]; }
        UserHandle = "CHandle";

SetDatabase(DrawingNoResult,0,j30,ObArea,"Solid","BA",CADHandleList10,TranVaule,CEName);
        break;
      case 7:     //if say NO
       MessageBox(CADWHandle ,"Select the Projected Area you have interest","Select",MB_OK);
       for(int i2 = 0;i2<Count;i2++)
        {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
         CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
         for(int h = 0;h<j30;h++)
          {CADHandle2 = CADHandleList10->Strings[h];
           if(CADHandle1 == CADHandle2)
             {SelectonScreen1.OleProcedure("Highlight",true);
              CADprompt= "Identify Projected Area you're interesting? N-not and continue /Y-
Yes: ";
              UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
              if(UserInput == "y"||UserInput == "Y")
                {ObArea = SBoxArea[2][h];
                 UserHandle = CADHandleList10->Strings[h];
                 SetDatabase1(DrawingNoResult,0,1,ObArea,"Solid","BA",UserHandle,CEName);
                 i2 = Count+2;
                 h=j30+2; }
              if(UserInput == "n"||UserInput == "N")
                {SelectonScreen1.OleProcedure("Highlight",false); } } } }
       break;
     }                       //End of Switch
    }                          //End of if(SObBoxArea..)
   delete CADHandleList10;
 }   //End of if(j30..)
//------------------------- SPLine j31 Single ---------------------------------------
//----------------------- SObArea[5][j31]  CADHandleList11 -------------------------
  if(j31==1)
   {ObArea = SObArea[5][0];
    UserHandle = CADHandleList11->Strings[0];
    SetDatabase1(DrawingNoResult,0,1,ObArea,"SPLine","A",UserHandle,CEName);}
//------------------------- SPLine j31 Multiple -------------------------------------
//----------------------- SObArea[5][j31]  CADHandleList11 -------------------------
  if(j31>1)
   {if(SObArea[5][1]>0)
```

```
{switch(MessageBox(CADWHandle ,"Got more than one Area, Add them each
   other?","Query",MB_YESNO))
  {case 6:      //if say Yes
     ObArea = 0;
     for(int n = 0;n<j31;n++)
      {ObArea+= SObArea[5][n];
       TranVaule[n] = SObArea[5][n]; }
     UserHandle = "CHandle";

SetDatabase(DrawingNoResult,0,j31,ObArea,"SPLine","A",CADHandleList11,TranVaule,CEName);
     break;
   case 7:      // if say No
     MessageBox(CADWHandle ,"Select the Area you have interest","Select",MB_OK);
     for(int i2 = 0;i2<Count;i2++)
      {SelectonScreen1 = PCADSelect.OleFunction("Item",i2);
       CADHandle1 = SelectonScreen1.OlePropertyGet("Handle");
       for(int h = 0;h<j31;h++)
         {CADHandle2 = CADHandleList11->Strings[h];
          if(CADHandle1 == CADHandle2 )
           {SelectonScreen1.OleProcedure("Highlight",true);
            CADprompt= "Identify the Area you're interesting? N-not and continue /Y-Yes:
";

            UserInput = CADUtility.OleFunction("GetString",0,CADprompt);
            if(UserInput == "y"||UserInput == "Y")
              {ObArea = SObArea[5][h];
               UserHandle = CADHandleList11->Strings[h];
               SetDatabase1(DrawingNoResult,0,1,ObArea,"SPLine","A",UserHandle,CEName);
               i2 = Count+2;
               h=j31+2;  }
            if(UserInput == "n"||UserInput == "N")
              SelectonScreen1.OleProcedure("Highlight",false); } } }
     break;
     }                          //End of switch
 }                              //End of if(SObAre..)
  delete CADHandleList11;
 }                                //End of if(j31..)
//-------------------- finish the part of this function design --------------------
}   //End of Search Function definition


//============================================================
// the function that used for get angle
//============================================================
double __fastcall TCADForm::GetAngle(Variant V1,Variant V2)
{ double Angle;
  double k1 = fabs(GetSlope(V1));
  double k2 = fabs(GetSlope(V2));
  if(k1==k2)
  {return 0;}
```

```
  if(1+k1*k2>1.05E-10)
   {if(k1==200000&&k2<0.00001)
     {return 90;}
    if(k1==200000&&k2<0.00001)
     {return 90;}
     if(k1*k2==0)
      { if( k1 == 200000)
          {Angle = 90-180/3.1415926*atan(k2);}
        if( k1 <0.00001)
          {Angle = 180/3.1415926*atan(k2);}

        if(k2 == 200000)
          {Angle =90-180/3.1415926*atan(k1);}
        if(k2<0.00001)
          {Angle =180/3.1415926*atan(k1);}
      }
    else {Angle = 180/3.1415926*atan( (k2-k1)/(1+k1*k2));}   }
  else
    {Angle = 90;}

 if(Angle<0)
 {Angle=-Angle;}
 if(Angle>180)
 {Angle-=180;}
switch(MessageBox(CADWHandle ,"Is it a obtus angle?","Query",MB_YESNO))
      {case 6:      //if say Yes
       Angle = 180-Angle;
       break;
       case 7:
       break;
       }

 return Angle;
}

//==============================================================
  // the function to get the slope of a line
//==============================================================
double TCADForm::GetSlope(Variant LineInput)
{
  Variant Par1 = LineInput.OlePropertyGet("StartPoint");
  Variant Par2 = LineInput.OlePropertyGet("EndPoint");
  double x1=Par1.GetElement(0);
  double y1=Par1.GetElement(1);
  double x2=Par2.GetElement(0);
  double y2=Par2.GetElement(1);
  if(x1!=x2)
   {return (y2-y1)/(x2-x1);}
  else
```

```
   {return 200000; }
}

//==============================================================
//                Get the Disance of parallel lines
//==============================================================
double TCADForm::GetPDistance(Variant LineInput1,Variant LineInput2){
    Variant Par11 = LineInput1.OlePropertyGet("StartPoint");
    Variant Par12 = LineInput1.OlePropertyGet("EndPoint");
    Variant Par21 = LineInput2.OlePropertyGet("StartPoint");
    Variant Par22 = LineInput2.OlePropertyGet("EndPoint");
    double x11,y11,x12,y12,x21,y21,x22,y22;
    double A,B,C,DResult;
    DResult = 0;
    x11 = Par11.GetElement(0);
    y11 = Par11.GetElement(1);
    x12 = Par12.GetElement(0);
    y12 = Par12.GetElement(1);
    x21 = Par21.GetElement(0);
    y21 = Par21.GetElement(1);
    x22 = Par22.GetElement(0);
    y22 = Par22.GetElement(1);
    if(x11==x12)
    { if(x21==x22)
      {DResult = abs(y11-y12);}
      else
     {MessageBox(CADWHandle ,"these two lines are not parallel","Warning",MB_OK);
      ShowWindow(CADWHandle,SW_SHOWMINIMIZED);
      ShowWindow(AppWHandle,SW_HIDE);
      ShowWindow(AppWHandle,SW_RESTORE); }    }
    else
    { A = (y22-y21)/(x22-x21);
      B = -1;
      C = y21-A*x21;
      DResult = abs(A*x11-y11+C)/sqrt(A*A+B*B);
      if(y11==y12)
       {if(y21==y22)
       {DResult = abs(x21-x11);}
      else
       {MessageBox(CADWHandle ,"these two lines are not parallel","Warning",MB_OK);}
       }
    }
   return DResult;
 } //End of define the GetPDitance


//==============================================================
//           Get the Area of the Box
//==============================================================
```

```
double TCADForm::GetBoxArea(Variant Cart1, int option){
     double x1,x2,y1,y2,z1,z2;
     double getObArea = 0;
     int j = Cart1.ArrayDimCount();
     for( int i = 0;i<j+1;i+=2)
     {x1 = Cart1.GetElement(i);
      y1 = Cart1.GetElement(i+1);
      z1 = Cart1.GetElement(i+2);
      x2 = Cart1.GetElement(i+3);
      y2 = Cart1.GetElement(i+4);
      z2 = Cart1.GetElement(i+5);
      double getArea;
      if(option==1)
      {getArea = abs((x1-x2)*(y1-y2));}
      if(option==2)
      {getArea = abs((x1-x2)*(z1-z2));}
      if(option==3)
      {getArea = abs((z1-z2)*(y1-y2));}
      getObArea = getObArea + getArea;}
     return getObArea;
   }


//==============================================================
//           Get total Length Function
//==============================================================
double TCADForm::GetTotalLength(Variant Cart1) {
     double x1,x2,y1,y2,z1,z2;
     double Distance = 0;
     int j = Cart1.ArrayDimCount();
     for( int i = 0;i<j+1;i+=2)
     { x1 = Cart1.GetElement(i);
       y1 = Cart1.GetElement(i+1);
       z1 = Cart1.GetElement(i+2);
       x2 = Cart1.GetElement(i+3);
       y2 = Cart1.GetElement(i+4);
       z2 = Cart1.GetElement(i+5);
       double Segment = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2));
       Distance = Distance + Segment;
     }
     return Distance;
}

//==============================================================
//                   Get the Project Area
//==============================================================
  double TCADForm::GetArea(Variant CPar1,Variant CPar2)
  {
    double x1=CPar1.GetElement(0);
```

```
    double y1=CPar1.GetElement(1);
    double x2=CPar2.GetElement(0);
    double y2=CPar2.GetElement(1);
    return abs((x1-x2)*(y1-y2));
   }

//======================================================
//                            Get the Length
//======================================================
double TCADForm::GetLength(Variant CPar1,Variant CPar2)
{ double x1=CPar1.GetElement(0);
  double y1=CPar1.GetElement(1);
  double z1=CPar1.GetElement(2);
  double x2=CPar2.GetElement(0);
  double y2=CPar2.GetElement(1);
  double z2=CPar2.GetElement(2);
           return sqrt( (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2) );
          }
```

C.2 FUNCTIONS IN UPDATE MODE

```
double     __fastcall     TFrmCADUpdate::ValueUpdate(AnsiString     PPUserHandle,double
PAutoCADUValue,AnsiString PCADType)
{ Variant UpdateSelection;
  AnsiString PUserHandle;
  if(PPUserHandle.LastDelimiter('%')>0)
   {  int SLength =  PPUserHandle.LastDelimiter('%');
      PUserHandle = PPUserHandle.Delete(1,SLength); }
  else
   { PUserHandle = PPUserHandle;}
 //this section is used for get the Handle without the '%'
  if(PUserHandle.Pos("CHandle")>0)
   {         // if it is the composite Handle
    for(int ListNum=0;ListNum<CADListcount;ListNum++)
     {
      TableUpdate->Next();       //go to subHandle4
      AnsiString UHandleLocal = TableUpdate->FieldByName("Handle")->AsString;
      AnsiString ULocalName = TableUpdate->FieldByName("FinalName")->AsString;
      for (int i=0;i<CADUCount;i++)
       { UpdateSelection = CADUpdateSelect.OleFunction("Item",i);
         String DrawingHandle = UpdateSelection.OlePropertyGet("Handle");
         if(DrawingHandle == UHandleLocal)
          {double CADULoValue = GetCADData(UpdateSelection,ULocalName);
                          // use gobal variable here
           PAutoCADUValue += CADULoValue;  //CADUpdateValue is gobal variable
```

```
            i = CADUCount;
            }           }
        return PAutoCADUValue;
        }          //End of for(int ListNum=..
    }
 //=== End of the compiste handle object that contain many handles in the DB ==
    else        //if it isn't a composite entity
     {for(int i=0;i<CADUCount;i++)
       {UpdateSelection = CADUpdateSelect.OleFunction("Item",i);
        String DrawingHandle = UpdateSelection.OlePropertyGet("Handle");
        ProgressBar1->StepIt();

        if(DrawingHandle == PUserHandle)
         {PAutoCADUValue = GetCADData(UpdateSelection,PCADType);
          i=CADUCount+1;    }
       }
     } //End of else
    return PAutoCADUValue;
 }      // End of definition of ValueUpdate function
//----------------------------------------------------------------------------------

void __fastcall TFrmCADUpdate::BtnClearDBClick(TObject *Sender)
{       //this is for clear the database have the same drawing No
 FrmCADUpdate->Height = 320;
 FrmCADUpdate->Width  = 273;
 ListBDB->Clear();
 EdtDNo->Text = "";
 Screen->Cursor = crHourGlass;
 PnlClearDB->Visible = true;
 PnlClearDB->BringToFront();

  AnsiString AutoCADNo;
 if(!TableUpdate->Active)
      TableUpdate->Open();
 TableUpdate->First();
 TStringList* DNumber = new TStringList();
 DNumber->Add("Drawing No :");
 Screen->Cursor = crHourGlass;
 int count = 0;
 while(!TableUpdate->Eof)
   {String DrawNo = TableUpdate->FieldByName("DrawingNo")->Value;
    if(DNumber->Strings[count]!=DrawNo)
     {DNumber->Add(DrawNo);
      count++; }
    TableUpdate->Next();
     }
 int j = DNumber->Count;
 for(int i=0;i<DNumber->Count;i++)
  {ListBDB->Items->Add(DNumber->Strings[i]);}
```

```
 ListBDB->MultiSelect = false;
 delete DNumber;
 Screen->Cursor = crDefault;
}


void __stdcall TFrmCADUpdate::USetDatabase(AnsiString PDrawingNo,int PType,int Pm,
                       double PFValue,AnsiString PName,AnsiString PFName,
                       AnsiString PObHandle,AnsiString PCEName)

{
     if(!TableUpdate->Active)
       TableUpdate->Open();
     int NumRecords=TableUpdate->RecordCount;
     AnsiString InputHandle;
     if(PObHandle.Pos("%")<=0)
       {InputHandle = PObHandle;}
     else
       {InputHandle = IntToStr(NumRecords)+"%"+PObHandle;}
      TableUpdate->Edit();
     try{TableUpdate->FieldByName("Handle")->Value= InputHandle;}
     catch(...)
     {switch(MessageBox(CADWHandle,"this  entity  has  in  the  database  already,do  you
want continue anyway?",
                       "Query",MB_YESNO))
     { case 6:
//if say Yes
       TableUpdate->FieldByName("Handle")->Value = PObHandle+"-2";
       break;
       case 7:
//if say No

       break; }
       }
     TableUpdate->FieldByName("DrawingNo")->Value = PDrawingNo;
     TableUpdate->FieldByName("SelectType")->Value = PType;
     TableUpdate->FieldByName("FName")->Value = PFName;
//such as A,L etc.
     TableUpdate->FieldByName("FinalValue")->Value = PFValue;
//the value that CE need
     TableUpdate->FieldByName("Name")->Value = PName;
     TableUpdate->FieldByName("MasterID")->Value = 1;           //indicate only one
Handle
     TableUpdate->FieldByName("CEDescription")->Value = PCEName;
     TableUpdate->Post();
     TableUpdate->Close();
}
```

```
double        __fastcall    TFrmCADUpdate::SearchANandD(Variant    PSelectSet,AnsiString
PHandle,AnsiString PType)
{ Variant UpdateSelection;
  AnsiString PUserHandle;
  if(PHandle.LastDelimiter('%')>0)
   { int SLength =  PHandle.LastDelimiter('%');
     PUserHandle = PHandle.Delete(1,SLength); }
  else
   { PUserHandle = PHandle;}

  double PAutoCADUValue;
  if(PUserHandle.Pos("$")>0)
   { int Pos = PUserHandle.Pos("$");
     AnsiString AnHandle1 = PUserHandle.SubString(1,Pos-1);
     PUserHandle.Delete(1,Pos);
     AnsiString AnHandle2 = PUserHandle;
     if(PType=="An"||PType=="D")
     { int CADUCount = PSelectSet.OlePropertyGet("Count");
       Variant Line1,Line2;
       int SearchCount = 0;
       String DrawingHandle;
       for (int i=0;i<CADUCount;i++)
       { UpdateSelection = PSelectSet.OleFunction("Item",i);
         DrawingHandle = UpdateSelection.OlePropertyGet("Handle");
         if(DrawingHandle == AnHandle1)
          {Line1 = UpdateSelection;
           SearchCount+=1;}
         if(DrawingHandle == AnHandle2)
          {Line2 = UpdateSelection;
           SearchCount+=1;}
         if(SearchCount ==2)
         {i = CADUCount;}
         ProgressBar1->StepIt();
       }
       if(PType=="An")
        {PAutoCADUValue = CADForm->GetAngle(Line1,Line2);}
        if(PType=="D")
        {PAutoCADUValue = CADForm->GetPDistance(Line1,Line2);}
       }
   }
  ProgressBar1->Visible = false;
  return PAutoCADUValue;
}
```